

AFIT/DS/ENS/98-1

19980924 062

AN INTEGER PROGRAM DECOMPOSITION  
APPROACH TO COMBAT PLANNING

DISSERTATION  
John C. Van Hove  
Captain, USAF

AFIT/DS/ENS/98-1

**DTIC QUALITY INSPECTED 1**

Approved for public release; distribution unlimited

The views expressed in this dissertation are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

AFIT/DS/ENS/98-1

AN INTEGER PROGRAM DECOMPOSITION APPROACH TO COMBAT PLANNING

DISSERTATION

Presented to the Faculty of the Graduate School of Engineering  
of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the  
Requirements for the Degree of  
Doctor of Philosophy in Operations Research

John C. Van Hove, B.S. in Operations Research, M.S. in Operations Research  
Captain, USAF

September, 1998


Approved for public release; distribution unlimited

AN INTEGER PROGRAM DECOMPOSITION APPROACH TO COMBAT PLANNING

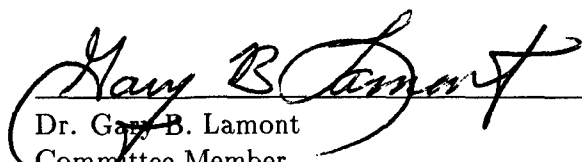
John C. Van Hove, B.S. in Operations Research, M.S. in Operations Research

Captain, USAF

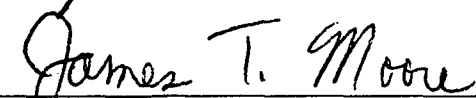
Approved:

  
\_\_\_\_\_  
Dr. Richard F. Deckro  
Committee Chairman

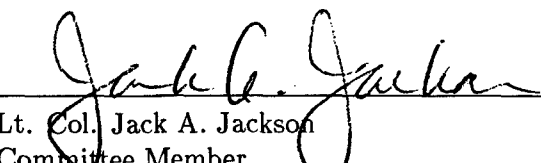
29 Aug 98  
Date

  
\_\_\_\_\_  
Dr. Gary B. Lamont  
Committee Member

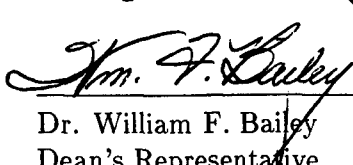
28 AUG 98  
Date

  
\_\_\_\_\_  
Lt. Col. James T. Moore  
Committee Member


29 Aug 98  
Date

  
\_\_\_\_\_  
Lt. Col. Jack A. Jackson  
Committee Member

29 Aug 98  
Date

  
\_\_\_\_\_  
Dr. William F. Bailey  
Dean's Representative

28 Aug 98  
Date

  
\_\_\_\_\_  
Robert A. Calico, Jr  
Dean



### *Acknowledgements*

The successful completion of this research would not have been possible without the support and guidance of many outstanding individuals. I am deeply thankful to my advisor, Dr. Richard F. Deckro, whose professional experience and technical expertise was invaluable throughout this research. In addition, I owe many thanks to my committee members, Dr. Gary B. Lamont, Lt. Col. James T. Moore, and Lt. Col. Jack A. Jackson, for their insightful advice and suggestions. I must also thank Dr. William F. Bailey, my dean's representative, for working with short suspenses as deadlines drew near, and First Lieutenant Kelly A. Herd for her much appreciated assistance with an administrative dilemma. Finally, I thank my family, especially my wife, Tammy, for keeping me sane during my five years at AFIT. This research is dedicated to them.

John C. Van Hove

## *Table of Contents*

	Page
Acknowledgements . . . . .	iii
List of Figures . . . . .	ix
List of Tables . . . . .	x
Abstract . . . . .	xii
 I. Introduction . . . . .	 1
1.1 Background . . . . .	1
1.2 Research Problem . . . . .	4
1.3 Research Objectives . . . . .	5
1.4 Assumptions . . . . .	6
1.5 Approach . . . . .	7
1.6 Summary . . . . .	8
 II. Literature Review . . . . .	 9
2.1 Project Management . . . . .	9
2.1.1 The Project Scheduling Problem. . . . .	9
2.1.2 The Resource Constrained Project Scheduling Problem. . . . .	12
2.1.3 The Generalized Resource Constrained Project Scheduling Problem. . . . .	13
2.1.4 Resource Constrained Crashing. . . . .	15
2.1.5 Activity Crashing. . . . .	16
2.2 Project Scheduling Algorithms . . . . .	20
2.2.1 Mathematical Programming. . . . .	21
2.2.2 Dynamic Programming. . . . .	23

	Page
2.2.3 Implicit Enumeration. . . . .	23
2.2.4 Branch and Bound. . . . .	24
2.3 Evolutionary Algorithms . . . . .	26
2.3.1 Evolutionary Algorithms for Project Scheduling. . . . .	26
2.3.2 Combining Evolutionary Algorithms with Deterministic Search. . . . .	28
2.4 Structure in Project Scheduling Models . . . . .	29
2.4.1 Dantzig-Wolfe Decomposition. . . . .	31
2.4.2 Sweeney-Murphy Decomposition. . . . .	32
2.5 Parallel Processing . . . . .	33
2.5.1 Parallel Evolutionary Algorithms. . . . .	34
2.5.2 Parallel Tree Search Algorithms. . . . .	36
2.6 Testing Project Scheduling Algorithms . . . . .	37
2.7 Summary . . . . .	38
III. Methodology . . . . .	39
3.1 Combat Planning Model Formulation . . . . .	39
3.1.1 Multi-modal Activities. . . . .	40
3.1.2 Doubly Constrained Resources. . . . .	42
3.1.3 Generalized Precedence Constraints. . . . .	45
3.1.4 Complete Model Formulation. . . . .	48
3.2 Combat Planning Solution Methods . . . . .	50
3.2.1 Subproblem Solution Methodology. . . . .	50
3.2.2 Decomposition Methodology. . . . .	51
3.2.3 Heuristic Methodology. . . . .	52
3.2.4 Test and Evaluation. . . . .	53
3.3 Summary . . . . .	54

	Page
IV. Subproblem Solution Algorithm . . . . .	55
4.1 Extending Sprecher . . . . .	55
4.1.1 Convergence. . . . .	56
4.1.2 Preprocessing. . . . .	62
4.1.3 The Basic Algorithm. . . . .	64
4.1.4 Acceleration Schemes. . . . .	68
4.1.5 Testing. . . . .	71
4.1.6 Complexity. . . . .	81
4.2 Summary . . . . .	84
V. A Decomposition Algorithm . . . . .	86
5.1 Sweeney-Murphy Decomposition . . . . .	86
5.1.1 The Sweeney-Murphy Algorithm. . . . .	88
5.1.2 Sweeney-Murphy Convergence. . . . .	90
5.2 Implementing Sweeney-Murphy Decomposition . . . . .	92
5.2.1 Finding the $k$ -Best Subproblem Solutions. . . . .	92
5.2.2 Setting and Resetting the Value of $k_i$ . . . . .	95
5.2.3 Finding Lagrangian Multipliers. . . . .	99
5.2.4 Solving the Sweeney-Murphy Master Problem. . . . .	103
5.3 Testing the Sweeney-Murphy Implementation . . . . .	104
5.4 Summary . . . . .	111
VI. An Evolutionary Algorithm Approach . . . . .	112
6.1 The Evolutionary Algorithm . . . . .	112
6.1.1 Basic Approach. . . . .	113
6.1.2 Individuals. . . . .	114
6.1.3 Operators. . . . .	116
6.2 Testing the Evolutionary Algorithm . . . . .	119

	Page
6.2.1 Configuring the Evolutionary Algorithm. . . . .	119
6.2.2 Comparing the Evolutionary Algorithm to Sweeney-Murphy. .	124
6.3 A Hybrid Approach . . . . .	125
6.4 Summary . . . . .	129
VII. Case Study . . . . .	130
7.1 Problem Generation . . . . .	130
7.1.1 Scenario. . . . .	130
7.1.2 Activities and Resources. . . . .	133
7.1.3 Precedence. . . . .	137
7.2 Sample Problem Results . . . . .	142
7.3 Summary . . . . .	147
VIII. Conclusions and Recommendations . . . . .	148
8.1 Research . . . . .	148
8.2 Contributions . . . . .	151
8.3 Recommendations . . . . .	152
8.4 Summary . . . . .	155
Appendix A. Setting $k_i$ Values . . . . .	156
Appendix B. Case Study Problem Data . . . . .	159
B.1 Target Nomination List . . . . .	159
B.2 Available Assets . . . . .	173
B.3 Mission Component Durations . . . . .	174
B.4 Precedence Networks . . . . .	187
B.5 Lag Values for Generalized Precedence . . . . .	190
B.6 The Optimal Solution . . . . .	215
Bibliography . . . . .	220

	Page
Vita . . . . .	227

## *List of Figures*

Figure		Page
1.	Combat Planning Spectrum . . . . .	2
2.	Data Managed by APS . . . . .	3
3.	Generalized Precedence Constraint Sets . . . . .	14
4.	Block-angular Structure . . . . .	29
5.	The Components of Mission Duration . . . . .	41
6.	A Feasible Utilization of Sorties . . . . .	44
7.	A Scenario for Generalized Precedence . . . . .	46
8.	Determining the Minimum Lag Value . . . . .	47
9.	The Complete MMGRCPSP Model Formulation . . . . .	49
10.	MMGRCPSP Model Formulation . . . . .	59
11.	A Comparison of Two Alternate Optimal Solutions . . . . .	76
12.	Theoretical Complexity Comparison . . . . .	83
13.	A Flow Diagram of the Sweeney-Murphy Algorithm . . . . .	88
14.	Empirical Results for $k$ -Best Complexity . . . . .	97
15.	A Flow Diagram of Hartmann's Evolutionary Algorithm . . . . .	113
16.	Average Deviation versus Total Individuals . . . . .	122
17.	Stopping Criteria Threshold Test Results . . . . .	123
18.	Standard Versus Hybrid Decomposition . . . . .	129
19.	A Map for the Case Study Scenario . . . . .	131
20.	Types of Precedence . . . . .	139
21.	Precedence Networks . . . . .	141
22.	Sortie Flow . . . . .	145
23.	Distribution of Solution Time . . . . .	157

# *List of Tables*

Table		Page
1.	A Sample TNL Entry . . . . .	40
2.	MMGRCPSP Variables and Parameters . . . . .	49
3.	MMGRCPSP Model Definitions . . . . .	59
4.	Existence Theorem . . . . .	61
5.	Preprocessing Algorithm Definitions . . . . .	63
6.	Preprocessing Algorithm . . . . .	63
7.	Basic Extended Sprecher Algorithm . . . . .	65
8.	Optimality Theorem . . . . .	68
9.	Step 5 for the Accelerated Algorithm . . . . .	70
10.	Precedence Information for the Sample Problem . . . . .	74
11.	Execution Mode Information for the Sample Problem . . . . .	74
12.	<i>K</i> -Best Solution Set for the Sample Problem . . . . .	75
13.	Constant Parameters Used By Sprecher and Drexl . . . . .	77
14.	Variable Parameters Used By Sprecher and Drexl . . . . .	77
15.	Constant Parameters For the First Test Set . . . . .	78
16.	Variable Parameters For the First Test Set . . . . .	78
17.	Solution Times For the First Test Set . . . . .	79
18.	Solution Times For the Second Test Set . . . . .	80
19.	Complexity Parameters . . . . .	81
20.	A Block-Angular Integer Program . . . . .	86
21.	The Sweeney-Murphy Subproblems . . . . .	87
22.	The Sweeney-Murphy Master Problem . . . . .	88
23.	Sweeney-Murphy Algorithm . . . . .	89
24.	Sweeney-Murphy Optimality Theorem . . . . .	90
25.	<i>K</i> -Best Solutions Theorem . . . . .	94



Table		Page
26.	How $k$ Affects Sweeney-Murphy Complexity . . . . .	98
27.	Finding Lagrangian Multipliers . . . . .	102
28.	Master Problem Solution Algorithm . . . . .	105
29.	Fixed Parameters for the Sweeney-Murphy Test Problems . . . . .	106
30.	Sweeney-Murphy Test Results . . . . .	107
31.	Sweeney-Murphy Test Results Reorganized . . . . .	109
32.	More Sweeney-Murphy Test Results . . . . .	110
33.	Hartmann's Crossover Operator . . . . .	117
34.	Configuring Population Size and Generation Limit . . . . .	120
35.	Comparing the Performance of Decomposition and EA . . . . .	124
36.	Comparing Hybrid and Standard Decomposition Approaches . . . . .	128
37.	Sample TNL Entries . . . . .	134
38.	Problem Resource Information . . . . .	135
39.	Sample Duration Information . . . . .	137
40.	Sample Lag Values . . . . .	142
41.	Sample ATO Solution Data . . . . .	144
42.	Solution Options . . . . .	146

*Abstract*

Over the last two decades, our military forces have been working to incorporate the latest computer technology into the combat planning process. The earliest efforts use word processors, spreadsheets, and databases to organize planning data and to display high level summaries for commanders. Later efforts perform feasibility checks as missions are planned to insure that the necessary resources are available and that the assets requested are capable of meeting the assigned scheduling requirements. Some of the most recent computer planning tools have included the capability to automatically plan individual missions or groups of missions. These automated efforts have been heuristic in nature due to the time limitations inherent to real-time combat planning. The methodologies in this research offer effective optimal alternatives to the limited heuristics available in the current combat planning tools.

This research formulates and solves a new class of project scheduling problems with applications to both military and civilian planning. It is shown that the solution space for this class of problems may be reduced in order to improve the effectiveness of both optimal and heuristic solution methodologies. In addition, a general method for extending implicit enumeration algorithms to obtain  $k$ -best solution sets is developed. The reduced solution space and the general  $k$ -best solutions methodology are exploited to develop several efficient solution approaches for this new class of problems; an implicit enumeration algorithm, a decomposition approach, an evolutionary algorithm, and a hybrid decomposition approach. The applicability and flexibility of the methodology are demonstrated with a case study that focuses on the force level planning of combat missions for an air campaign. While the focus of the case study is combat planning, the concepts illustrated are applicable to the general field of program and project management.

# AN INTEGER PROGRAM DECOMPOSITION APPROACH TO COMBAT PLANNING

## *I. Introduction*

In the not too distant past, our military forces accomplished all aspects of combat planning with little more than laminated maps, grease pencils, and paper tablets. The past success of our armed forces in combat attests to the fact that such systems have worked. However, advanced technology is beginning to dominate modern combat, which is increasingly characterized by joint operations involving multiple services, and multiple nations, who must work closely together to employ advanced technologies to their fullest potential. As today's battlefield grows more complex and technology dependent, precision and coordination are increasingly important. In a world where conflict can happen at the speed of light, the grease board approach is no longer adequate.

Over the last several decades, specialized software has been introduced into the planning environment to aid our combat planners in performing their tasks more effectively. In certain cases, these software tools have attempted to completely automate some combat planning tasks. For the most part, this automation is accomplished with greedy heuristics, since the available optimal seeking algorithms have not been operationally feasible in the time-critical, short horizon, combat planning environment. As will be shown, the process of planning combat missions is very similar to the process of scheduling activities in a project and can be modeled as a variant of the classic project scheduling problem. An efficient algorithm for optimally solving this type of problem would offer a feasible alternative to the greedy heuristics currently in use.

### *1.1 Background*

Computers and specialized software tools have become a part of nearly all areas of combat planning in today's military [3, 11, 38, 39, 40, 41, 61, 65, 73, 77]. Figure 1 illustrates the spectrum of combat planning, showing the various levels of planning while also illustrating the joint and combined responsibilities of the planning function. The levels are shown as the depth of planning and include campaign level, force level, and unit level. The degree of 'jointness' is labeled as breadth of planning and shows how a military operation may be planned for a single service branch, multiple

service branches, or even a multi-national force. The profile depicted in Figure 1 illustrates how combat planning can range from a squadron commander planning the details of a strike against a single enemy target to the commander of a multi-national force setting the objectives for a major campaign.

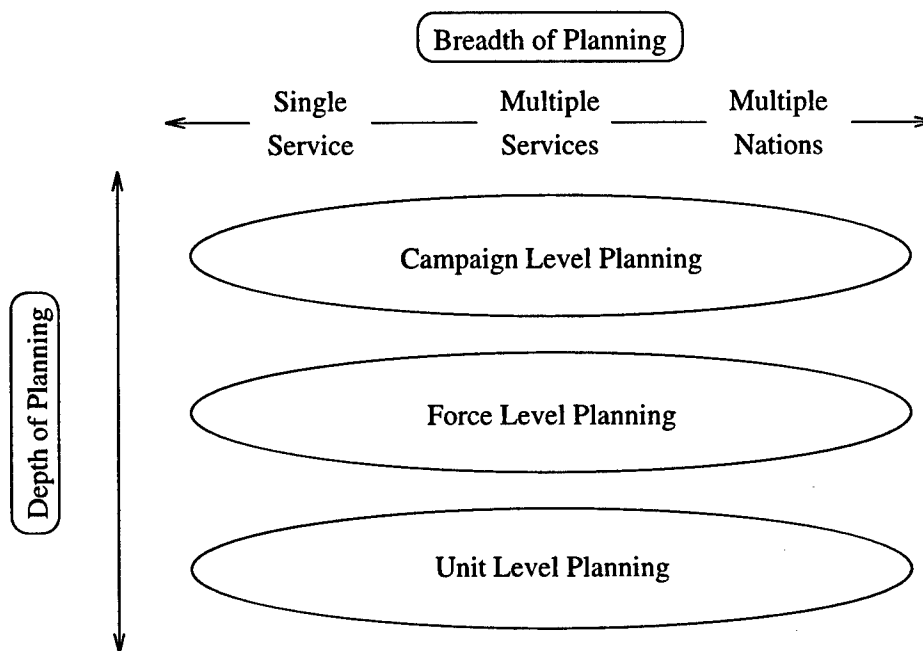


Figure 1 Combat Planning Spectrum

This research effort develops a general modeling approach that is applicable to the wide range of problems on the combat planning spectrum illustrated in Figure 1. To demonstrate the applicability of the approach, the research focuses on the force level planning of an air campaign and the automated tools that are used in this area. The Advanced Planning System (APS) is the most recent computer aided combat mission planning tool acquired for the Air Force's force level combat planners [39, 61]. Inputs to APS include a prioritized, weaponized Target Nomination List (TNL), commander's guidance, weather reports, reconnaissance requests, and the various logistical information that represents what resources are available for use in the next day's combat. The single output from APS is an Air Tasking Order (ATO), a document generated at the theater level that tasks assets to strike targets over a 24-hour period [41]. The variety of input and output data managed by APS is displayed by the diagram in Figure 2.

The concept of APS was to replace the laminated maps, grease pencils, and paper tablets in the combat plans shop with digitized maps, user friendly worksheets, and a computer database [39]. APS enforces coordination between the various planners, performs feasibility checks on planned missions, and makes a number of useful algorithms available to the combat planners. The ultimate goal is to make the ATO generation process more effective so that more targeting alternatives may be considered and the ATO generation cycle may be shortened.

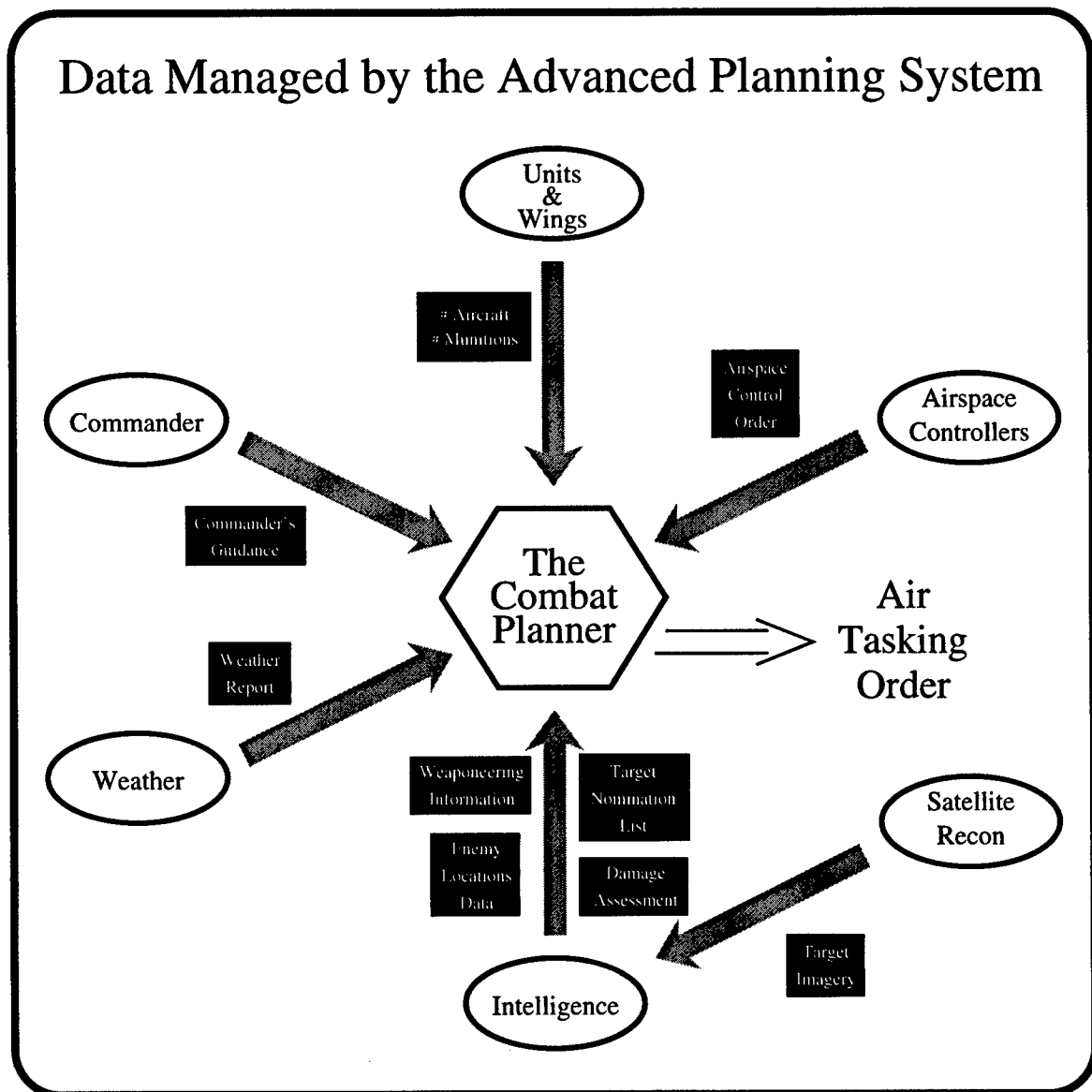


Figure 2 Data Managed by APS

The APS feature that is most important to this research is the heuristic driven capability to automatically plan combat missions. The current implementation allows the planner to select a list of targets from the TNL and pass them on to the auto-planner where resources are matched against targets to build mission lines for the ATO. The auto-planner uses a greedy heuristic that considers the selected targets one at a time in the order that the planner sends them. For each target, the auto-planner selects the best resources that are currently available, builds a mission with that resource/target pairing, and schedules it at the first available time in the ATO day. If there are no resources left to use against a target, a mission including that target is not planned.

The APS approach is only marginally useful in the planning process. Since the proposed missions are not necessarily an optimal allocation of available resources, the combat planner often has to make major adjustments to assure adequate missions are planned for all of the targets. This type of sub-optimal heuristic was selected over more time-intensive optimal algorithms because combat planners are often faced with a very short planning horizon when planning operations during an ongoing event [39, 41, 61].

## *1.2 Research Problem*

The research in this dissertation develops a methodology for performing optimal automatic combat mission planning in the limited time available to combat planners. As previously stated, the automatic mission planning problem can be thought of as a variant of the classic project scheduling problem. However, planning for a scenario of realistic size quickly becomes time prohibitive for any of the existing solvers for this type of problem [13, 36, 79, 92]. This effort develops a methodology that takes advantage of the special underlying structure in both project management and combat planning problems to improve the solution efficiency in terms of the time to find an optimal solution. In addition, the methodology and algorithms developed for the combat planning problem have direct application in the area of general program and project management.

### 1.3 Research Objectives

The methodology developed in this dissertation effort utilizes the Sweeney-Murphy method of integer program decomposition [84] to exploit the special structure of the problem. This research accomplishes the following specific objectives:

1. To illustrate the applicability of program and project management techniques to combat planning, the force level mission planning problem for an air campaign is examined and an appropriate mathematical programming model is developed in Chapter III.
  - (a) The model uses multi-modal activities to allow several different weaponizing options to be considered for every target.
  - (b) The model uses doubly constrained resources to limit the total number of aircraft and the total number of sorties available from each unit in a scenario.
  - (c) The model uses generalized precedence constraints to specify the timing required between certain missions in the ATO.
2. To provide a subproblem solution algorithm for a decomposition approach, an efficient method for solving small combat planning problem instances is identified in Chapter IV.
  - (a) The subproblem solution algorithm is extended from the fastest exact solution methods available in the literature for resource constrained project scheduling.
  - (b) The subproblem solution algorithm is the first of its kind to support both multi-modal activities and generalized precedence constraints.
  - (c) A general extension for implicit enumeration algorithms is developed to allow the subproblem solution algorithm to generate  $k$ -best solution sets.
3. To obtain optimal solutions to larger problem instances, a decomposition algorithm, based on Sweeney-Murphy decomposition, is developed and fine tuned for optimal performance on combat planning problems in Chapter V.
  - (a) The subproblem solution algorithm supports the Sweeney-Murphy requirement for  $k$ -best solution sets from each subproblem.

- (b) Effective strategies are developed for determining Lagrangian multipliers and values for  $k$  in the subproblems.
  - (c) A specialized algorithm is developed that applies the implicit enumeration extension to obtain  $k$ -best solution sets for the Sweeney-Murphy master problem.
4. To provide a solution alternative for problem instances too large for optimal solution methodologies, an evolutionary algorithm approach to the combat planning problem is developed in Chapter VI.
- (a) The evolutionary algorithm approach incorporates multi-modal activities, doubly constrained resources, generalized precedence constraints, and  $k$ -best solution sets.
  - (b) A hybrid decomposition approach is developed by combining the evolutionary algorithm with the Sweeney-Murphy decomposition methodology.
5. To demonstrate the applicability of the model and solution methodologies to the combat planning problem, a case study is conducted in Chapter VII.

#### 1.4 Assumptions

Development of the mathematical programming formulation and solution methodology for the combat planning model required the following set of assumptions:

1. It is acceptable to plan the attack missions first and then plan the missions that are necessary to provide support for those attack missions.
2. Problems are formulated from the data that serves as input to APS. The logistics considerations (human and hardware resource availability) have already been considered.
3. Mission duration for a given resource/target pairing can be approximated by a known, deterministic value.
4. Any resource/target pairing is acceptable if and only if there is an acceptable weaponeering option in the TNL for that resource against that target.
5. The airfields can accommodate whatever schedule the optimal solution dictates.



### 1.5 Approach

To increase the upper bound on problem size for which it is reasonable to obtain optimal solutions, the methodologies developed in this dissertation research use decomposition to exploit block-angular problem structure. First, the combat planning problem is formulated as a mathematical programming model with block-angular structure. The block-angular structure allows the problem to be decomposed into a number of semi-independent subproblems (divide and conquer) that have the same form as project scheduling problems with multi-modal activities and generalized precedence constraints.

The literature is rich with applications of, and solution techniques for, the classic project scheduling problem. The Critical Path Method (CPM) solves the classic project scheduling problem as a network flow problem, for which there are very efficient solution methods. Multi-modal activities, however, change the formulation significantly enough that CPM techniques can no longer be directly applied. Either further decomposition is employed to exploit the special network structure or a specialized implicit enumeration technique is used [79].

In either case, the solution technique must be modified to obtain  $k$ -best solution sets instead of a single optimal solution. This is necessary since the Sweeney-Murphy algorithm optimizes the main problem by exploring combinations of the  $k$ -best solutions to the subproblems. This research develops a general extension to implicit enumeration algorithms that allows an algorithm to provide the top  $k$  problem solutions instead of a single optimal solution. This implicit enumeration extension is applied to both the subproblem and master problem solution algorithms. While the  $k$ -best subproblem solutions are required by the decomposition method, the  $k$ -best master problem solutions are obtained to provide the benefit of near optimal solution alternatives.

Aside from  $k$ -best solution techniques, there are two other aspects of the Sweeney-Murphy algorithm that are very problem specific. The first is the method for choosing the value of  $k$  for each subproblem. In the few published applications of the Sweeney-Murphy algorithm, these values were selected arbitrarily, even though the computational time for the decomposition may be greatly affected by the choice of  $k$ . The other problem specific aspect of the Sweeney-Murphy algorithm is the selection of values for the Lagrangian multipliers that are used in the objective function of each

subproblem. These aspects allow the Sweeney-Murphy algorithm to be fine-tuned for the specific combat planning problem formulation.

After the decomposition algorithm is implemented, a heuristic solution methodology using evolutionary algorithms is developed. For larger problem instances, it is still too computationally expensive to use the decomposition algorithm to find an optimal solution. The heuristic approach may offer a “near” optimal solution in a fraction of the time. In addition, the heuristic approach is combined with the exact algorithm to form a hybrid approach that is able to obtain an optimal solution for larger problem instances in less time than the standard exact algorithm.

The solution methodologies developed in this dissertation research are tested on a wide range of sample problems. The performance of these solution methodologies is compared with any methods from the literature that accomplish similar objectives. Additionally, the performance of each method is compared with the performance of all the other methods developed in this dissertation effort in order to illustrate the relative strengths of all methods. The decomposition approach is tested against the subproblem solution algorithm, the evolutionary algorithm is tested against the decomposition approach, and the hybrid approach is tested against the standard decomposition approach. After all solution approaches were developed and tested, the most effective exact algorithm was applied in a case study in order to demonstrate the applicability of the approach for combat planning.

## *1.6 Summary*

This chapter presented an overview of the focus of this dissertation. Chapter II presents a comprehensive review of the literature pertaining to project management, project scheduling algorithms, evolutionary algorithms, decomposition algorithms, and other topics related to the research in this dissertation. Chapter III presents the formulation for the combat planning problem and outlines the solution methodologies that are developed to solve this problem. Chapters IV, V, and VI describe the subproblem solution algorithm, the decomposition algorithm, and the evolutionary algorithm, respectively. In Chapter VII, a case study is performed to demonstrate the applicability of the approach to large combat planning problems. Chapter VIII provides a summary of the research, its significant contributions, and recommendations for future work.

## *II. Literature Review*

The overall thrust of this research is to 1) develop a mathematical programming formulation to model force level combat mission planning and 2) develop a solution methodology to effectively solve large problems of this class. A secondary goal of this research is to utilize the methodology to solve the corresponding class of program and project management scheduling problems. To these ends, several types of project scheduling models and solution methods from the literature are reviewed. First, project scheduling algorithms are considered. Evolutionary algorithms for project scheduling are then covered. Finally, the possibility of using evolutionary algorithms and deterministic search in a combined approach is examined.

After discussing the various project scheduling models and solution techniques, several approaches for exploiting structure in both the actual problem and the solution methodology are explored. The model formulation possesses some network structure, as well as some elements of classic block-angular structure. For these reasons, several techniques that decompose the main problem into semi-independent subproblems are examined. Some parallel processing techniques are then presented that show promise in taking advantage of structure in the solution methodology.

### *2.1 Project Management*

For the purpose of this research, a project is defined as a finite set of activities that must be accomplished according to certain precedence requirements between activity pairs [92]. Depending on the specific project, certain assumptions can be made concerning activity duration, resource consumption, and resource availability. Project management deals with the planning and scheduling of large projects or, in the context of this effort, air tasking orders. The objective of project scheduling problems is normally to minimize either the overall project cost or completion time.

*2.1.1 The Project Scheduling Problem.* The simplest project scheduling problem (PSP) considers only activity duration and precedence information, assuming adequate resources for any schedule, and generally has an objective of minimizing total project completion time. Model (*KM*) below is Ahuja's formulation of Kelley's original model of the classic project scheduling problem. [1, 45].

$$(KM) \quad \text{Minimize } x_n - x_1 \quad (1)$$

$$\text{Subject to } x_j - x_i \geq \tau_i \quad \forall (i, j) \in P \quad (2)$$

$$x_i \text{ unrestricted} \quad \forall i \in A \quad (3)$$

In this formulation, the set  $A$  is a set of all  $n$  activities in the project while  $P$  is the set of all  $(i, j)$  activity pairs where activity  $i$  must be completed before activity  $j$  can begin. The parameter  $\tau_i$  represents the duration of activity  $i$ . The decision variable  $x_i$  represents the earliest start time of activity  $i$ . The objective function, given by Equation (1), is formulated as the difference between the earliest start times of the first and last activities, the project makespan. This assumes that there is only one activity with no predecessors (activity 1) and only one activity with no successors (activity  $n$ ). If this is not true, dummy source and sink activities may be added to the model to satisfy the assumption. The precedence constraints, given by Equation (2), restrict activity start times to insure no activity may start before its predecessors are completed.

This model can also be formulated as a longest path network flow problem and specialized Critical Path Method (CPM) techniques may be used to find the minimum project completion time more efficiently than standard linear program (LP) solvers [1]. For this formulation, it is convenient to develop a network representation of the project scheduling problem. In this case, an *activity on the node* representation has been used. In an activity on node representation, each activity is represented by a node in the network and an arc is directed between nodes  $i$  and  $j$  if and only if job  $i$  must be completed before job  $j$  can begin. There is a parameter  $\tau_i$ , the duration of activity  $i$ , associated with every node in the network.

The network representation must have only one source node and one sink node. If there is only one activity with no predecessors, then its node is the network's source node. Otherwise, a dummy source node may be added and zero-length arcs are directed from the dummy to each activity node without a predecessor. Likewise, if there is a single activity with no successors, then its node is the network's sink node. If not, a dummy sink node may be added and a zero-length arc is directed from each node without a successor to the dummy sink node. This network representation of the

project scheduling problem is a weighted digraph with one source node and one sink node, where all component arcs are part of at least one source to sink path.

This structure can be used to find the shortest project completion time by applying a network flow technique to determine the length of the longest path. Each path in the network represents a set of activities that must be accomplished sequentially to satisfy precedence requirements. To allow for all precedence relations in a network path to be satisfied, the length of a path is equal to the sum of the durations of every activity in that path. The project completion time can be no smaller than the length of any network path; therefore, the length of the longest network path defines the shortest possible project completion time. The longest network path (or paths) is referred to as a *critical path(s)*. Model (NF) below is Ahuja's LP formulation of the longest path network flow problem [1].

$$(NF) \quad \text{Maximize} \quad \sum_{(i,j) \in P} \tau_i y_{ij} \quad (4)$$

$$\text{Subject to} \quad \sum_{i \forall (i,j) \in P} y_{ij} - \sum_{k \forall (j,k) \in P} y_{jk} = \begin{cases} -1 & \text{if } j = 1 \\ 0 & \forall j \in A - \{1, n\} \\ 1 & \text{if } j = n \end{cases} \quad (5)$$

$$y_{ij} \geq 0 \quad \forall (i,j) \in P \quad (6)$$

The sets and parameters are the same as those used in the first model formulation, but the variables are different. Each  $y_{ij}$  decision variable represents the quantity of flow along the arc between activity node  $i$  and activity node  $j$ . This model flows one unit of flow from source to sink in the network representation of the project scheduling problem. The objective function, given by Equation (4), moves this one unit of flow across the longest source to sink path by maximizing the sum of the weights of the component arcs of this path. In effect, finding the string of sequential activities that has the longest duration, the critical path. The constraints in Equation (5) maintain conservation of flow while those in Equation (6) restrict flow to be strictly non-negative.

The optimal objective function value of each model is equal to the shortest possible completion time. In fact, each model can be shown to be the dual of the other [44]. Since model (NF) is a pure network flow model, its constraint matrix is totally unimodular and thus so is the constraint matrix

of model ( $KM$ ), its dual. With the totally unimodular constraint matrices, the solution obtained by a standard LP solver is integer as long as the right-hand-side (RHS) values in the model are integer [44]. This can be a very useful property if the start times of activities are restricted (or scaled) to integer values in a given project scheduling problem.

*2.1.2 The Resource Constrained Project Scheduling Problem.* The classic project scheduling model does not account for limited project resources. Therefore, a solution that is optimal for the classic model may be infeasible to a model that considers available project resources. The Resource Constrained Project Scheduling Problem (RCPSP) model formulation results from the expansion of the classic model formulation to handle limited resources. The formulation of the RCPSP presented here is adapted from Pritsker, Watters, and Wolfe [74]. Of the various RCPSP formulations in the literature, this one requires the fewest binary variables. This formulation is stated with many of the same parameters as the previous models, but it requires quite a few additional variable types which are defined below.

Constants:

- $A$  the set of all activities
- $K$  the set of all resources
- $P$  the set of all activity precedence pairs
- $g$  the project deadline
- $\tau_i$  the duration of activity  $i$
- $e_i$  the earliest completion time for activity  $i$
- $l_i$  the latest completion time for activity  $i$
- $r_{ik}$  the amount of resource  $k$  required by activity  $i$
- $R_{jk}$  the amount of resource  $k$  available in period  $j$

Variables:

- $x_{it}$  = 1 if activity  $i$  finishes in period  $t$ ; 0 otherwise

$$\text{Minimize } \sum_{t=e_n}^{l_n} tx_{nt} - \sum_{t=e_1}^{l_1} tx_{1t} \quad (7)$$

$$\text{Subject to } \sum_{t=e_n}^{l_n} tx_{nt} - \sum_{t=e_i}^{l_i} tx_{it} \geq \tau_n \quad \forall (i, n) \in P \quad (8)$$

$$\sum_{i \in A} \sum_{t=j}^{j+\tau_i-1} r_{ik} x_{it} \leq R_{jk} \quad \forall k \in K \text{ and } j = 1, \dots, g \quad (9)$$

$$\sum_{t=e_i}^{l_i} x_{it} = 1 \quad \forall i \in A \quad (10)$$

$$x_{it} \in \{0, 1\} \quad \forall i \in A \text{ and } t = 1, \dots, g \quad (11)$$

The most profound difference between this RCPSP formulation and the classic project scheduling problem formulation previously presented lies in the decision variables. In the RCPSP formulation, it is necessary to have a series of binary decision variables for each activity to account for per period resource consumption. This is a significant increase over the number of decision variables in the classic project scheduling model, which only required a single continuous decision variable for each activity. In addition, the speed of LP solvers is sacrificed by the need to use binary variables to accurately model this situation.

The objective function, given by Equation (7), minimizes project completion time, but its statement is more complex than that of the PSP. Similarly, the precedence constraints stated in Equation (8) are a more complex statement of the PSP precedence constraints. The constraints given in Equations (9) and (10) do not appear in the PSP. The constraints expressed in Equation (9) maintain resource consumption within limits for each resource type available in each period while the constraints in Equation (10) simply state that each activity can only be completed in one of the possible time periods.

**2.1.3 The Generalized Resource Constrained Project Scheduling Problem.** In the formulation just given for the RCPSP, activity timing requirements are enforced by Equation (8). These constraints specify that activity  $i$  must be completed before activity  $j$  can begin for all activity precedence pairs  $(i, j)$ . Not all project scheduling problems have activity precedence requirements that conform to this standard. To deal with this, the Generalized Resource Constrained Project

Scheduling Problem (GRCPSP) uses four different constraint formulations to model all possible precedence relations. The four sets of activity precedence relations are defined below and illustrated in Figure 3.

- $H_1$  start-start activity pair relations with a lag of  $SS_{ij}$
- $H_2$  start-finish activity pair relations with a lag of  $SF_{ij}$
- $H_3$  finish-start activity pair relations with a lag of  $FS_{ij}$
- $H_4$  finish-finish activity pair relations with a lag of  $FF_{ij}$

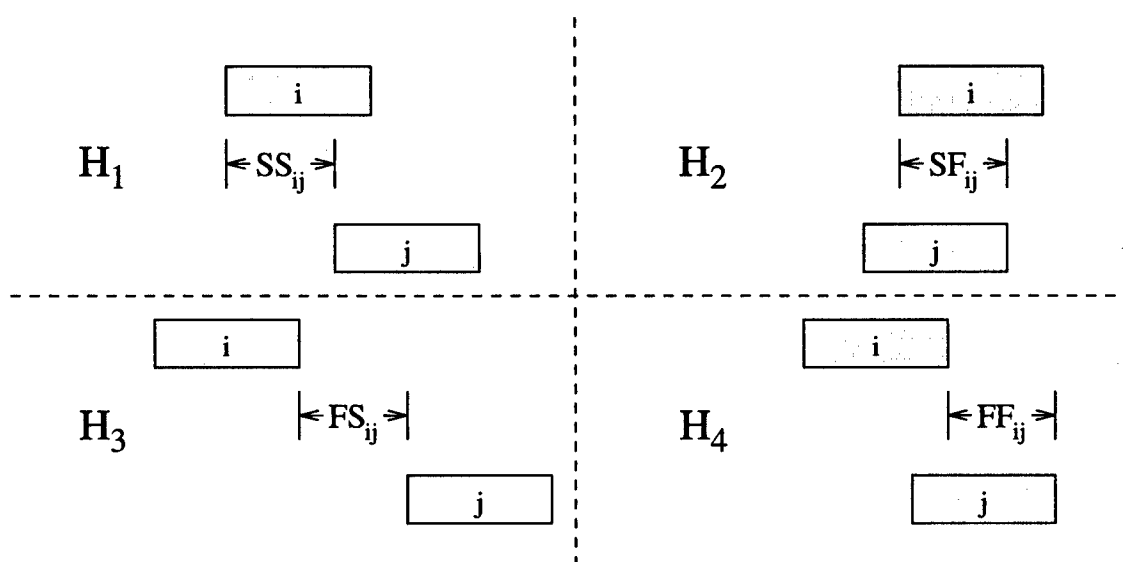


Figure 3 Generalized Precedence Constraint Sets

Figure 3 illustrates the various types of activity precedence conditions in a manner similar to the traditional Gantt chart used in scheduling theory. A Gantt chart represents activities as blocks of time with lengths corresponding to the duration of the activity. These blocks are arranged on a graph where the horizontal axis represents time and the vertical axis represents multiple machines or some other mechanism for processing activities in parallel. The four sets of activity precedence conditions are depicted similarly in Figure 3. For example, the simple precedence arrangement used by all of the models presented thus far could be expressed by a type  $H_1$  constraint with  $SS_{ij} = \tau_i$ . In the  $H_1$  portion of the figure, there are blocks representing activities  $i$  and  $j$  and an illustration of the constraint that enforces a minimum time of  $SS_{ij}$  between the start of activity  $i$  and the start of activity  $j$ .



The other three types of precedence conditions are enforced similarly. The difference is in which activity endpoints are used. A minimum amount of lag time is enforced between the start of the predecessor and the completion of the successor for type  $H_2$  precedence, between the completion of the predecessor and the start of the successor for type  $H_3$  precedence, and between the completion of the predecessor and the completion of the successor for type  $H_4$  precedence. A GRCPSP model adapted from [13] applies these four types of generalized precedence relations as follows.

$$\text{Minimize } y_n - y_1 \quad (12)$$

$$\text{Subject to } \sum_{t=e_i}^{l_i} tx_{it} = y_i \quad \forall i \in A \quad (13)$$

$$y_j - y_i \geq SS_{ij} + \tau_j - \tau_i \quad \forall (i, j) \in H_1 \quad (14)$$

$$y_j - y_i \geq SF_{ij} - \tau_i \quad \forall (i, j) \in H_2 \quad (15)$$

$$y_j - y_i \geq FS_{ij} + \tau_j \quad \forall (i, j) \in H_3 \quad (16)$$

$$y_j - y_i \geq FF_{ij} \quad \forall (i, j) \in H_4 \quad (17)$$

$$\sum_{i \in A} \sum_{t=j}^{j+\tau_i-1} r_{ik} x_{it} \leq R_{jk} \quad \forall k \in K \text{ and } j = 1, \dots, g \quad (18)$$

$$\sum_{t=e_i}^{l_i} x_{it} = 1 \quad \forall i \in A \quad (19)$$

$$x_{it} \in \{0, 1\} \quad \forall i \in A \text{ and } t = 1, \dots, g \quad (20)$$

$$y_i \geq 0 \quad \forall i \in A \quad (21)$$

The difference between this formulation and the RCPSP formulation given previously is the ability to express a wider variety of precedence requirements which requires additional variables and constraints. The constraints in Equation (13) define continuous decision variables in terms of the original, binary decision variables. These variables,  $y_i$ , represent the completion time of activity  $i$  and serve to simplify the statement of the precedence constraints. Every possible requirement for timing among activities in a project scheduling problem is covered by the set of precedence constraints given by Equations (14), (15), (16), and (17).

**2.1.4 Resource Constrained Crashing.** In many cases, the overall project completion time in a RCPSP is dictated by resource availability and not by the duration of the critical path. For

this reason, it may be cost effective in a project to purchase additional resources in order to shorten the project completion time. This concept is referred to as *resource constrained crashing*. The resource constrained crashing formulation given below is an extension of the Pritsker, Watters, and Wolfe RCPSP model [74] originally proposed by Deckro and Hebert [19].

$$\text{Minimize } \sum_{k \in K} \sum_{j=1}^g c_{jk} w_{jk} \quad (22)$$

$$\text{Subject to } \sum_{t=e_n}^{l_n} tx_{nt} - \sum_{t=e_m}^{l_m} tx_{mt} \geq \tau_n \quad \forall (m, n) \in P \quad (23)$$

$$\sum_{i \in A} \sum_{t=j}^{j+\tau_i-1} r_{ik} x_{it} - w_{jk} \leq R_{jk} \quad \forall k \in K \text{ and } j = 1, \dots, g \quad (24)$$

$$0 \leq w_{jk} \leq \mu_{jk} \quad \forall k \in K \text{ and } j = 1, \dots, g \quad (25)$$

$$\sum_{t=e_i}^{l_i} x_{it} = 1 \quad \forall i \in A \quad (26)$$

$$x_{it} \in \{0, 1\} \quad \forall i \in A \text{ and } t = 1, \dots, g \quad (27)$$

This model makes the assumption that if the project cannot be completed by the deadline with the current resource availability, more resources may be made available in certain periods in order to meet the deadline. To model the potential for additional resources, a new set of variables is added to the original RCPSP model. The new variables,  $w_{jk}$  represent the quantity of additional resource  $k$  made available during period  $j$ . The additional quantities of resources that can be made available are restricted according to Equation (25).

The possibility of additional resources affects both the objective function and the resource constraints of the original RCPSP model formulation. The objective of the original model was to minimize the overall project completion time; now the goal, given in Equation (22), is to meet the project deadline at minimal additional resource cost. Other objectives are possible. The new resource constraints are given in Equation (24) and reflect the possible additional resources available every period.

**2.1.5 Activity Crashing.** Resource constrained crashing provides a mechanism for shortening overall project completion time by increasing the availability of certain critical resources in

key periods to compress as much of the slack time out of a schedule as possible. Activity crashing provides a similar means of shortening project completion time, but tightens the schedule by cutting activity duration time instead of slack time between activities. This is accomplished by assigning an explicit or implicit cost function to activity duration (an activity can be completed more rapidly, but it may require more resources). The objective of models that use this type of crashing is typically either to meet a specific deadline at a minimal crashing cost or to optimize a cost function that balances the positive value of completing a project early against the crashing costs incurred to make an early completion time possible.

The activity crashing models presented here assume that if the project deadline cannot be met, additional resources can be applied to shorten the project completion time. The goal here, just as in the resource crashing models, is to meet the project deadline at a minimal crashing cost. This type of crashing was originally presented by Kelley [45] who adapted CPM techniques to allow for shortening project completion time in this manner. The first activity crashing model presented here is adapted from a Foldes and Soumis [27] model that uses a very generic statement of crashing costs. This model is a simple extension of the classic project scheduling problem formulation ( $KM$ ) given previously in Section 2.1.1.

$$\text{Minimize } \sum_{i \in A} c_i(\tau_i) \quad (28)$$

$$\text{Subject to } x_j - x_i - \tau_i \geq 0 \quad \forall (i, j) \in P \quad (29)$$

$$x_n - x_1 \leq g \quad (30)$$

$$a_i \leq \tau_i \leq b_i \quad \forall i \in A \quad (31)$$

$$x_i \text{ unrestricted} \quad \forall i \in A \quad (32)$$

In this formulation, the activity durations,  $\tau_i$ , have changed from parameters to variables reflecting the possibility of crashing activity duration. The amount of crashing for any activity is bounded by Equation (31), the durations are moved to the left-hand-side of the precedence constraints in Equation (29) since they are no longer constant, and the constraint in Equation (30) is added to reflect the deadline for project completion. The objective of the model, to meet the deadline at a minimum crashing cost, is expressed in Equation (28) in very general terms. The

function  $c_i(\tau_i)$  is used to represent the cost of completing activity  $i$  with a duration of  $\tau_i$ . In general, it is expected that  $c_i(\tau_i)$  should increase as  $\tau_i$  decreases, but the exact form of the cost function is left to the discretion of the modeler.

Wiley, Deckro, and Jackson [94] suggest a more specific cost function. This model includes both crashing and extending of activities and uses linear cost functions for both.

$$\text{Minimize } \sum_{i \in A} K_i y_i - \sum_{i \in A} E_i z_i \quad (33)$$

$$\text{Subject to } x_j - x_i + y_i - z_i \geq \tau_i \quad \forall (i, j) \in P \quad (34)$$

$$x_n - x_1 \leq g \quad (35)$$

$$y_i \leq \mathcal{Y}_i \quad \forall i \in A \quad (36)$$

$$z_i \leq \mathcal{Z}_i \quad \forall i \in A \quad (37)$$

$$y_i, z_i \geq 0 \quad \forall i \in A \quad (38)$$

$$x_i \text{ unrestricted} \quad \forall i \in A \quad (39)$$

In this formulation,  $K_i$  is the cost of reducing the duration of activity  $i$  by one time period, while  $E_i$  is the benefit of extending its duration by one time period. Linear costs are assumed to preserve the LP form of the model. The activity durations,  $\tau_i$ , are once again constants, but the variability is now included using the crashing and extending variables  $y_i$  and  $z_i$ . These variables affect the formulation of the precedence constraints in Equation (34) and the objective function in Equation (33). As in the previous model, the duration of each activity is bounded. This is accomplished with the upper bound constraints given by Equations (36) and (37) where  $\mathcal{Y}_i$  represents the maximum amount that the duration of activity  $i$  may be reduced through crashing and  $\mathcal{Z}_i$  represents the maximum amount that the duration of activity  $i$  may be increased through extending.

The assumption of linear cost functions, however convenient, may not be totally realistic in some cases. If non-linear cost functions must be used, the model formulation changes from a LP to a non-linear program (NLP) and solution complexity increases. Non-linear cost functions can be approximated with piecewise linear functions, but for the purposes of this study, continuous non-linear cost functions are estimated by a set of discrete points. This is convenient since it is often the case that resources are only available in discrete units (workers or machines for example)

and discrete resources dictate a discrete cost function. This is the concept behind the Multi-Modal Resource Constrained Project Scheduling Problem (MMRCPSP), the last model presented in this section.

In the MMRCPSP model, activities can be completed in one of a number of possible execution modes. The resource consumption and activity duration vary by mode. The multi-modal model presented below is based on a model expressed by Boctor [7] and closely resembles the RCPSP model formulation examined previously.

- $A$  the set of all activities
- $K$  the set of all resources
- $P$  the set of all activity precedence pairs
- $M_i$  the set of all execution modes for activity  $i$
- $\tau_{im}$  the duration of activity  $i$  in mode  $m$
- $e_i$  the earliest completion time for activity  $i$
- $l_i$  the latest completion time for activity  $i$
- $r_{imk}$  the amount of resource  $k$  for activity  $i$  in mode  $m$
- $R_{jk}$  the amount of resource  $k$  available in period  $j$
- $x_{imt} = 1$  if activity  $i$  finishes in period  $t$  using mode  $m$

$$\text{Minimize } \sum_{m \in M_n} \sum_{t=e_n}^{l_n} tx_{nmt} - \sum_{m \in M_1} \sum_{t=e_1}^{l_1} tx_{1mt} \quad (40)$$

$$\text{Subject to } \sum_{m \in M_j} \sum_{t=e_j}^{l_j} (t - \tau_{jm}) x_{jmt} - \sum_{m \in M_i} \sum_{t=e_i}^{l_i} tx_{imt} \geq 0 \quad \forall (i, j) \in P \quad (41)$$

$$\sum_{i \in A} \sum_{m \in M_i} \sum_{t=j}^{j+\tau_i-1} r_{imk} x_{imt} \leq R_{jk} \quad \forall (j, k) \quad (42)$$

$$\sum_{m \in M_i} \sum_{t=e_i}^{l_i} x_{imt} = 1 \quad \forall i \in A \quad (43)$$

$$x_{imt} \in \{0, 1\} \quad \forall (i, m, t) \quad (44)$$

The addition of multiple activity execution modes to the general RCPSP model forces a change in notation. The binary decision variables,  $x_{imt}$ , take a value of one if and only if activity  $i$  is to

be executed in mode  $m$  and completed in period  $t$ . Unlike the other crashing models presented, the crashing cost is not explicitly accounted for in the objective function. The objective function in Equation (40) has reverted back to modeling the original goal of minimizing overall project completion time. The cost is implied by the trade-off of scarce resources between the project's activities (resources are applied where they make the largest impact on project completion time).

## 2.2 *Project Scheduling Algorithms*

The classic project scheduling problem presented in Section 2.1.1 has a dual that is a longest path problem which, as a pure network flow problem, can be solved with algorithms that are more efficient than a standard simplex approach [44]. One such efficient algorithm is the classic CPM approach, which is thoroughly covered by Wiest and Levy [92]. In the CPM approach, the classic project scheduling problem is solved by way of a labeling algorithm that makes both a forward and backward pass through the activities.

On the forward pass, the nodes are considered in order of precedence (a node cannot be considered until all of its predecessors have been considered) and an earliest start time is assigned as the maximum of the earliest finish times of its predecessors. If an activity has no predecessors, its earliest start time is set to the project start time, usually zero. The earliest finish time for an activity is calculated as the sum of its earliest start time and its duration. After the forward pass, the project completion time is available. The project completion time is equal to the maximum of the earliest finish times of all project activities and some specified horizon limit (the maximum planning period).

The backward pass of the CPM algorithm is used to provide windows on the possible start times of all activities if the project completion time found in the forward pass is to be met. On the backward pass, the activities are considered in reverse order of precedence and latest start and finish times are assigned to all activities. The latest finish time is found as the minimum of the latest start times of an activities successors. If the activity has no successors, its latest finish time is set to the project completion time. The latest start time for an activity is calculated by subtracting the activity duration from its latest finish time.

In these two passes, the labeling algorithm computes the earliest project completion time and assigns windows for the feasible start and finish times for each activity in the project. Wiest and Levy [92] give the mathematical programming formulation of the classic project scheduling problem and point out that the linear programming approach is less efficient but offers the advantages of duality and sensitivity analysis.

*2.2.1 Mathematical Programming.* The mathematical program for the classic project scheduling problem is a pure network flow model. Large problems of this kind can be solved with a standard LP solver. With the RCPSP, however, the model is a complex integer program where even relatively small projects have very large integer programming formulations. Wiest and Levy [92:page 131] give the following example of the daunting size of a RCPSP formulation that does not even include any additional variables or constraints for crashing activities or resources. This example uses a model developed by Bowman [8], which is perhaps the worst case when sheer model size is considered. Others have developed formulations that reduce the number of variables and constraints, but the resulting models are still intimidating for problems of realistic size.

For example, a project with 50 jobs having 100 precedence relationships and requiring four different resources over a time span of 30 days has 1,500 variables and over 6,500 constraints (not counting slack variables).

Although the sheer size of integer programming formulations of RCPSP is far from encouraging, the literature is rich with a wide variety of formulations for the many variants of this problem. The RCPSP model formulation presented in Section 2.1.2 was based on the work of Pritsker, Waters, and Wolfe [74] and has been extended by Talbot [87]. In this formulation, for every activity there is a binary decision variable for every period in which it could feasibly be completed. In Section 2.1.4 an extension of this formulation by Deckro and Hebert [19] was presented that adds resource crashing to the model. This extension allows additional resources to be obtained at a cost in order to improve the project completion time.

Deckro and Hebert also extended a RCPSP model formulation from Bowman [8]. In Bowman's formulation, each activity has one binary decision variable for every time period from the earliest possible activity start time to the latest possible activity completion time. Inside of this window, a string of variables take on values of one to represent the time when the activity is being worked

on in the proposed schedule. Constraints are used to insure that all of these unitary variables are sequential to prohibit activity splitting. One strength of this model is that these constraints may be left out if activity splitting is allowed. Deckro and Hebert [19] add a crashing variable to this model formulation for each activity that allows activity duration to be reduced at a linear cost within bounds similar to the Wiley, Deckro, and Jackson [94] model presented in Section 2.1.5.

In Herroelen's review of integer programming approaches to the RCPSP [36] he describes a binary integer programming model formulation originally proposed by Riester and Schwinn [75] and then simplified by Franke [28]. This model uses two different sets of binary decision variables; one set to determine the schedule of the activities and another to determine the completion time of the project. Franke also presents two alternative formulations that do not require the division of the planning time-frame into fixed periods. In these formulations, the binary variables are used to denote whether or not one activity must be completed before some other may begin.

The binary integer programming formulations of the RCPSP given by Bowman [8], Talbot [87], and Pritsker, Watters, and Wolfe [74] have more recently been extended by Boctor [7] and Sprecher [79] to handle activities with multiple execution modes. Boctor's formulation is the MMRCPPSP model formulation presented in Section 2.1.5 and is almost identical to the model Sprecher uses. These model formulations use a set of binary decision variables for each activity, one variable for every feasible activity/mode/schedule combination. In a feasible solution, exactly one variable from the set of variables for a given activity has a value of one. All other variables in the set have a value of zero. The single nonzero variable in the set defines the schedule and execution mode of the activity in question. The multiple activity execution modes allow a sort of activity crashing since the shorter modes require more project resources.

On page 21 a quote from Wiest and Levy [92] was given to illustrate the large size of the integer programming formulation of even a relatively small RCPSP. The example contained no additional variables or constraints for activity crashing, resource crashing, or multiple activity execution modes. For this reason, every RCPSP model presented in this section has an associated integer programming formulation at least as large as, and in most cases even larger than the Bowman model. Empirical evidence from the literature shows that solving the RCPSP with straight-forward integer programming methods becomes computationally inefficient for all but the smallest of prob-



lems [13, 36, 79, 92]. Sprecher [79:page 9] states the following about his computational experience with solving integer programming models of this type.

The linear programming formulation suggests to employ a linear programming based algorithm, e.g. Branch and Bound with LP-relaxation, but the computational results ...show that *even small problems are intractable by this approach.*

**2.2.2 Dynamic Programming.** Several early attempts at solving the RCPSP with a dynamic programming approach can be found in the literature, but the curse of dimensionality keeps these approaches from being an efficient method for problems of interesting size [13, 36]. Carruthers and Battersby [10] present a dynamic programming approach to a special, disjunctive class of the RCPSP, but they run into dimensionality problems even with this restricted problem class. Petrovic [71] looked at the application of dynamic programming methods to the resource leveling problem. This problem is a variant of the RCPSP where the goal is to even out resource consumption over all periods in the project by minimizing the sum of the squared deviations from the mean resource use per period. Petrovic's approach also falls victim to the curse of dimensionality. The discouraging results from past work and the lack of current effort leads to the conclusion that dynamic programming is not currently a promising approach for solving the RCPSP.

**2.2.3 Implicit Enumeration.** There are many examples in the literature of implicit enumeration procedures, particularly branch and bound techniques, for the RCPSP [13, 36, 70]. Balas [4] formulates a RCPSP and then shows that the problem is similar to a machine-sequencing problem. He goes on to show that these similar problems can both be reduced to finding an optimal set of arcs in a disjunctive graph with stability conditions. Using this reduced problem, Balas develops an implicit enumeration algorithm to solve the RCPSP, but gives no empirical data to gauge the efficiency of the algorithm.

Davis and Heidorn [18] present an implicit enumeration procedure for the RCPSP based on techniques originally developed for solving the assembly line balancing problem. To use this approach, each activity must be split into a number of unit duration activities equal to the duration of the original activity. The advantages of this approach are that activity preemption can be modeled and activities with variation in resource usage can be handled (all of the unit duration activities for

an original activity do not need to have the same resource requirements). The main disadvantage of the approach is large memory storage requirements.

Talbot [86, 88] gives another implicit enumeration procedure for the RCPSP that utilizes a formulation of the problem with general integer variables instead of the usual binary programming formulations. This formulation requires much less memory than the other approaches. His procedure evaluates all possible combinations of finish times for all of the activities in the project. His major contribution was the concept of a cut for eliminating the explicit enumeration of inferior combinations early in the enumeration scheme. This concept led to stronger fathoming rules than available with a general implicit enumeration algorithm.

*2.2.4 Branch and Bound.* The more recent, and more efficient, implicit enumeration procedures for the RCPSP found in the literature are generally branch and bound type approaches [13, 70]. Stinson [83] developed a branch and bound procedure for the RCPSP that used a skiptracking technique to navigate the solution tree. Each node in the solution tree represented one possible precedence and resource feasible assignment of a subset of the project activities. Any constraints on activities not in the current subset are ignored at that node in the tree. Stinson's procedure was tested extensively on both the RCPSP and the job-shop scheduling problem with notable success [70].

Patterson [70] conducts a comparison of Stinson's branch and bound procedure with the implicit enumeration procedures of Davis [18] and Talbot [86, 88] mentioned in Section 2.2.3. Patterson collected 110 test problems that contained every RCPSP test problem found in the literature up to 1984. All of these problems consisted of a single project of 7–50 activities requiring 1–3 resource types. Activity splitting was not allowed, resources were available at a constant level per period, and activities required resources in constant amounts for their entire duration. Patterson ran each procedure on all 110 of the test problems and evaluated them on the basis of the number of problems solved to optimality, the average solution time, and the number of problems where each method was the fastest. While Talbot's procedure had the smallest memory requirement and Davis's procedure is the fastest on problems with few precedence-feasible options, Stinson's procedure was significantly better in all three areas of evaluation.

Christofides, Alvares-Valdes, and Tamarit [14] present a branch and bound procedure that uses a depth-first tree search technique, CAT, to solve the RCPSP. Each node in the solution tree represents a semi-active partial solution that is both precedence and resource feasible. A semi-active partial solution is a schedule for a subset of the project activities in which none of the scheduled activities could be scheduled earlier without violating a precedence or resource constraint. Activities are added to the partial schedule until there is a resource conflict, at which point branching occurs. A branch is added to the node of the current partial solution for each possible conflict resolution. The authors claim that the CAT procedure compares favorably with the Stinson procedure, but when used to solve the 110 Patterson problems, Demeulemeester [24] found that it solved fewer problems to optimality and had a much higher average solution time.

An extension of the CAT procedure, DH, was given by Demeulemeester and Herroelen [23] using the same semi-active partial solution nodes. The improvements of the DH procedure over the CAT procedure are guaranteed convergence to the optimal solution and improved solution time. These improvements were achieved with the use of better selection rules for adding activities to partial schedules and better dominance rules for pruning the solution tree. In 1995, Demeulemeester stated that the DH procedure was the fastest exact solution method for solving the the RCPSP [13]. The DH procedure has also been extended to handle the Resource Availability Cost Problem (RACP) [21] and the Preemptive Resource Constrained Project Scheduling Problem (PRCPSP) [25].

The final branch and bound procedure to be covered in this section was developed by Sprecher [79] to solve the variant of the RCPSP where the activities have multiple execution modes (MM-RCPSP). This procedure builds semi-active partial schedules like the DH procedure, but requires unique rules for selecting activity/mode combinations to add to the partial schedule and for determining dominance to prune the solution tree. Kolisch and Frase [47] suggest new rules to tighten the window of feasible activity start times, new dominance rules, and a new feasibility rule all to increase the efficiency of Sprecher's branch and bound procedure. A more detailed discussion of Sprecher's algorithm is presented in Chapter IV.

### 2.3 Evolutionary Algorithms

Evolutionary algorithms are heuristic optimization techniques that implement a stochastic search procedure based on the principle of evolution in nature. This heuristic approach has been successfully applied to a wide variety of problem areas and has gained a strong following in the optimization community [29, 30].

The basis of the classic evolutionary algorithm is a population of individuals, represented by fixed-length binary sequences, that represent possible solutions to an optimization problem. The quality of these individuals is gauged by a function that measures the fitness of any potential solution with respect to the objective of the optimization problem at hand. Based on their relative fitness, individuals are selected for reproduction and produce offspring.

A crossover operator is used to produce one or two children by recombining the binary sequences of two parents. The children may undergo mutation, where there is a small probability of random changes to their binary sequences. The quality of the children is evaluated based on the fitness function. The population for the next generation is formed by either replacing all of the parents with the most fit children or by selecting the most fit of both parents and children. Evolution continues in this manner, continually generating new populations of higher quality individuals, until some stopping criteria has been met.

Many variants of this general evolutionary algorithm approach have been developed and applied: using non-binary sequences, different recombination operators, different mutation operators, multiple populations, and local search techniques [29]. These more general evolutionary based heuristics are sometimes referred to as evolutionary programs [62]. In this research, all heuristic approaches based on the principle of evolution are referred to as evolutionary algorithms.

*2.3.1 Evolutionary Algorithms for Project Scheduling.* There are examples of evolutionary algorithms applied to scheduling problems throughout the literature. Gen and Cheng [29] provide a good survey of the various evolutionary algorithm techniques that have been applied to flow-shop sequencing, job-shop scheduling, and machine scheduling problems. Sprecher [79] has shown that the job-shop scheduling problem is a special case of the RCPSP, but in general there have been very few evolutionary algorithms proposed for the RCPSP.

In their chapter on evolutionary algorithms for job-shop scheduling, Gen and Cheng point out that evolutionary algorithms are not well suited for fine-tuning structures that are very close to optimal solutions [29]. They go on to suggest that for scheduling problems it is important to incorporate conventional heuristics into evolutionary algorithms to make the evolutionary algorithm more competitive. Several efficient heuristics have been proposed for the MMRCPPSP that would be candidates for use in an evolutionary algorithm approach [7, 68, 69, 78]. The possibilities for incorporating one or more of these heuristics into an evolutionary algorithm include the following:

- design the heuristic into the crossover operator
- design the heuristic into the mutation operator
- use the heuristic in the generation of the initial population
- use the heuristic in the fitness function to map individuals to schedules
- use the heuristic for local search to improve offspring

Lee and Kim [58] were the first to apply evolutionary algorithms to the RCPSP. In their work, Lee and Kim apply three new heuristic approaches to the RCPSP: simulated annealing, tabu search, and evolutionary algorithms. They compare the results of these three approaches with results from six other heuristic methods. The authors claim that the advantages of their heuristics are that 1) only feasible solutions are generated, 2) no preliminary problem information is required, 3) any objective can be handled, and 4) the notion is simple and easy to implement. Lee and Kim apply their three approaches and six other heuristic approaches to the 110 Patterson test problems [70] as well as some other randomly generated problems. The set of randomly generated problems was created to include problems with varying levels of number of activities, number of resources, and tightness of resource constraints. Their empirical results show the simulated annealing, tabu search, and evolutionary algorithm approaches consistently outperformed the other six heuristic methods.

Hartmann went one step farther than Lee and Kim by applying evolutionary algorithms to the MMRCPPSP [35]. Hartmann points out that the only other application of evolutionary algorithms to this problem area comes from Özdamar [66] and is based on priority rule encoding. Hartmann's approach represents individual population members as a precedence feasible activity sequence and a mode assignment. This genetic string maps to a schedule by considering the activities sequentially

in the order indicated and each activity is scheduled in the assigned mode at the earliest feasible time. Fitness is measured according to the makespan of the resulting schedule.

An interesting feature of Hartmann's approach is its use of local search to improve the fitness of individuals by applying problem specific knowledge, a concept introduced by Grefenstette [32]. Hartmann's mapping of genetic strings to schedules results in feasible schedules, but by applying problem specific knowledge, it is often possible to improve those schedules with very little computation. To do this Hartmann uses a technique developed by Sprecher [79] called multi-mode left shift. The technique calls for the activities to be considered sequentially and for each activity it is determined if it would be resource feasible to use a mode with a shorter duration. By changing the execution mode of some activities, the makespan can sometimes be shortened and the fitness of the potential solution is improved.

Hartmann tests his approach against both the latest heuristics and branch and bound algorithms. The branch and bound approach he uses is a truncated version of the method proposed by Sprecher and Drexel [80, 81]. This truncated approach was presented as a method for using branch and bound as a heuristic when computation time for an optimal solution to larger problems is prohibitive. Based upon the criteria of average deviation from optimal, Hartmann's empirical results show that his evolutionary algorithm outperformed each of the other heuristic methods.

*2.3.2 Combining Evolutionary Algorithms with Deterministic Search.* Lamont, Gates and Brown present several ideas on how to increase optimization efficiency by combining the strengths of deterministic search and evolutionary algorithms [55]. The approach consists of increasing the efficiency of one method by seeding it with promising solutions from another. The three main possibilities for applying this approach are as follows.

1. Seed the initial population of an evolutionary algorithm with solutions obtained in the early stages of a deterministic search algorithm
2. Use the best feasible solution from the latest generation of an evolutionary algorithm as the starting bound for a deterministic search algorithm
3. Exchange promising solutions back and forth between an evolutionary algorithm and a deterministic search algorithm, possibly in parallel

The second possibility, using heuristically obtained solutions to provide a good initial bound for deterministic search algorithms, is a common optimization technique. Normally in a branch and bound algorithm, branches cannot be excluded from consideration (fathomed) until a feasible solution, which provides a bound, has been found. The advantage of obtaining feasible solutions externally is that fathoming can begin earlier in the search process. The better these initial feasible solutions are, the earlier fathoming can take place. Slowiński, Soniewicki and Weglarz provide an example of such a technique applied to the MMRCPSp [78]. The authors present a decision support system that uses parallel priority rules and simulated annealing to heuristically generate good feasible solutions to a MMRCPSp with multiple objectives. If the decision maker is not satisfied with the quality of the proposed solutions, the solutions are passed on as bounds to a branch and bound algorithm so as to find better solutions.

#### 2.4 Structure in Project Scheduling Models

This section examines the possibility of exploiting the structure of problems that exhibit the block-angular structure illustrated in Figure 4. The illustration in Figure 4 represents the constraint

$A_1$	$A_2$	$A_3$
$B_1$		
	$B_2$	
		$B_3$

Figure 4 Block-angular Structure

matrix of a problem whose variables may be partitioned into discrete sets such that certain blocks of constraints, represented by the matrices  $B_i$ , are unique to one set of variables. The constraints represented by the  $A_i$  matrices are referred to as linking constraints.

The block-angular structure illustrated in Figure 4 is found in the multi-project scheduling problem (MPSP). The MPSP is the natural extension to problems where one large program is

composed of several semi-independent projects. By semi-independent, it is meant that most of the problem's constraints affect only a single project, but there are some constraints, called linking constraints, that affect more than one project. These might be common resources or requirements. Such a structure follows naturally from the classic work breakdown structure. Consider the following general mathematical programming formulation.

$$\text{Minimize } \mathbf{c}\mathbf{x} \quad (45)$$

$$\text{Subject to } \mathbf{A}\mathbf{x} \geq \mathbf{b} \quad (46)$$

$$\mathbf{x} \geq \mathbf{0} \quad (47)$$

If the problem modeled by this general formulation had block-angular structure, then its constraint matrix would have the following form.

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 & \mathbf{A}_2 & \cdots & \mathbf{A}_p \\ \mathbf{B}_1 & & & \\ & \mathbf{B}_2 & & \\ & & \ddots & \\ & & & \mathbf{B}_p \end{bmatrix} \quad (48)$$

A problem with a constraint matrix of this form is said to have  $p$  block-angular structure since there are  $p$  distinct blocks [56]. The  $\mathbf{A}_i$  are matrices of coefficients for the linking constraints and the  $\mathbf{B}_i$  are matrices of coefficients for constraints unique to project  $i$ . The advantage of this problem structure is that large problems which may be computationally prohibitive to solve by direct methods can be decomposed into a number of subproblems of a more manageable size [56]. The general formulation expressed by Equations (45), (46), and (47) with a constraint matrix of the form in Equation (48) can be expanded as follows. The constraints in Equation (50) are the linking constraints and those in Equation (51) are the project unique constraints.



$$\text{Minimize } \sum_{i=1}^p \mathbf{c}_i \mathbf{x}_i \quad (49)$$

$$\text{Subject to } \sum_{i=1}^p \mathbf{A}_i \mathbf{x}_i \geq \mathbf{b}_0 \quad (50)$$

$$\mathbf{B}_i \mathbf{x}_i \geq \mathbf{b}_i \quad \text{for } i = 1, 2, \dots, p \quad (51)$$

$$\mathbf{x}_i \geq \mathbf{0} \quad \text{for } i = 1, 2, \dots, p \quad (52)$$

*2.4.1 Dantzig-Wolfe Decomposition.* Problems with block-angular structure can be solved with decomposition by resources. Techniques that use decomposition by resources fix or relax the linking constraints in order to arrive at independent subproblems. One classic method of decomposition by resources is the Dantzig-Wolfe decomposition principle for linear programs [17]. In Dantzig-Wolfe decomposition, the linking constraints are relaxed to yield an independent subproblem for each block, but they are included in a master problem that considers convex combinations of subproblem solutions. The dual variables associated with the linking constraints in the master problem are used to weight the objective functions of the subproblems in order to guide each subproblem's solution toward compliance with the linking constraints.

The Dantzig-Wolfe procedure consists of iteratively solving the subproblems, adding the new subproblem solutions to the master problem, solving the master problem, and deriving new subproblem objective functions based on the new dual variables of the master problem. The iterative procedure continues until the latest subproblem solutions can offer no improvement to the objective function of the master problem. At this point, the current master problem solution is optimal for the original problem. The optimal values for the original decision variables are obtained as convex combinations of the decision variables of the master problem.

Several authors have applied Dantzig-Wolfe type solution techniques to the variants of the MPSP [5, 91, 93, 94]. The main advantage of this approach is the reduced size of the mathematical programming formulations that must be solved, allowing faster solution of larger problems. Other benefits, pointed out by Baumol and Fabian [5], are the insights into cost tradeoffs obtained from the iterative subproblem solutions. The drawback to a Dantzig-Wolfe decomposition approach is that its application is restricted to linear programming models. The many integer and binary

programming model formulations of resource constrained problems cannot be solved with Dantzig-Wolfe decomposition.

*2.4.2 Sweeney-Murphy Decomposition.* Sweeney and Murphy [84] developed a method of decomposition for integer programming models that exhibit block-angular structure. Sweeney-Murphy decomposition is similar to Dantzig-Wolfe decomposition. The subproblems and the master problem have the same structure, but the variables are integer rather than continuous and therefore lack the luxury of dual variables unless the constraint matrices are totally unimodular. Without dual variables, the ability to guide the solutions of the subproblems towards compliance with the linking constraints is seriously weakened. In addition, the integer requirement does not allow for simple convex combinations of the solutions of a subproblem. Instead of a convex combination, the master problem must select one of the solutions for each of the subproblems.

In Sweeney-Murphy decomposition, a solution for the master problem, and also for the original problem, is the combination of one  $k$ -optimal solution from each of the subproblems. Instead of generating subproblem solutions iteratively, a set of the top  $k$  solutions (a  $k$ -best solution set) to each subproblem is generated in a single step of the procedure. The master problem is then solved, finding the best combination of these  $k$ -optimal subproblem solutions. Sweeney and Murphy developed an optimality test that either declares a solution optimal or points to specific subproblems for which the next  $k$  solutions must be obtained before optimality can be achieved. They also outline a procedure to place bounds on the optimal solution once a feasible solution is found. This allows the decomposition algorithm to be terminated if the current feasible solution is *close enough* to optimal.

There are three aspects of the Sweeney-Murphy procedure that are not fully developed in their presentation. First, since dual variables are not directly available, what are the best weights to use in the subproblem objective functions? Sweeney and Murphy suggest several possibilities, including using the dual variables from the LP relaxation of the master problem, but give no empirical study supporting any one weight selection technique for any specific problem class.

Another open issue is the selection of  $k$ , the size of the near-optimal solution sets. According to the Sweeney-Murphy algorithm, this can be a single, fixed value or a different value for each subproblem, but again, no guidelines are established. In their one empirical example, Sweeney and

Murphy arbitrarily set  $k$  to a common value for all of the subproblems, but they gave no reason for their choice [84]. Clearly, the value of this parameter can effect the efficiency of the entire procedure so it is important that some thought goes into its selection.

The last open issue is how to obtain the  $k$ -best solution sets of the subproblems. Sweeney and Murphy do not go into the method used in their example. The issue of determining the most efficient method for obtaining  $k$ -best solution sets is highly problem specific. There are several examples from the literature where algorithms have been developed to find the  $k$ -best solutions to a specific combinatorial optimization problem [9, 34, 57, 63]. It is likely that resolution of the other two open issues, selecting values for weights and  $k$ , are also very specific to the type of problem.

In a later effort, Sweeney and Murphy, apply their integer program decomposition technique to the specific problem area of multi-item scheduling [85]. The multi-item scheduling problem involves scheduling the production of many items over time. In this application, the subproblems have a unique formulation. Each subproblem is a single multiple choice (special ordered set) type constraint [90] and so it is relatively easy to obtain  $k$ -best subproblem solution sets. The authors demonstrate the usefulness of the proposed approach on three real applications.

In an article on resource constrained project crashing, Deckro and Hebert present a pair of models for crashing in resource constrained project scheduling problems [19]. They go on to suggest that there are possibilities for improved calculation through decomposition in each of these formulations. In another effort, Deckro, Winkofsky, Hebert, and Gagnon outline a procedure for using Sweeney-Murphy decomposition on the MPSP [20]. As proof of concept, the authors present empirical results for an illustrative example.

## 2.5 Parallel Processing

The following definitions are obtained from Kumar's introductory text on parallel computing [53]. When implementing an algorithm in a parallel environment, the goal is to maximize speedup. Define  $T_s$  as the runtime for the fastest sequential algorithm for a given problem and  $T_p$  as the runtime of a parallel algorithm for the same problem executed on  $p$  processors. The performance measure speedup is defined as the ratio of serial runtime to parallel runtime,  $S = T_s/T_p$ . Theoretically, the

upper bound on speedup is  $p$ . Speedup of  $p$  is referred to as linear speedup. If superlinear speedup is achieved (speedup in excess of  $p$ ), the sequential algorithm is not as efficient as it could be.

Another important measure of performance is efficiency. Efficiency is defined as the ratio of speedup to the number of processors used,  $E = S/p$ . Since the upper bound on speedup is  $p$ , the best efficiency that can be achieved is 1. This level of efficiency is typically difficult to achieve due to the additional communication requirements that occur as a result of accomplishing tasks of an algorithm on separate processors.

Since the goal in most block-angular decomposition techniques is to relax a few constraints in order to create independent subproblems, decomposition algorithms are good candidates for parallel implementation. There are several examples in the literature where LP decomposition techniques like Dantzig-Wolfe have been implemented in parallel [37, 76, 95]. However, there are no examples of a parallel integer program decomposition technique.

There are two types of parallelization possible when implementing an algorithm in a parallel environment, control parallelism and data parallelism. Control parallelism is breaking an algorithm down into tasks and completing tasks in parallel on separate processors. When a task consists of performing identical processing on many data items, data parallelism is assigning data elements to multiple processor to accomplish the identical processing in parallel.

Control parallelism applied to Sweeney-Murphy integer program decomposition would be straight forward. Each subproblem could be assigned to a separate processor in order to find the  $k$ -best solution sets in parallel. For this step of the algorithm, speedup would be very close to linear with no communication necessary and the only idle processor time would be caused by the difference in the size and complexity of the subproblems. Data parallelism, however, would require re-implementing the subproblem solution algorithm in parallel. Such a parallel implementation potentially involves both evolutionary algorithms and branch and bound algorithms. Parallel versions of these techniques are addressed in the following sections.

*2.5.1 Parallel Evolutionary Algorithms.* Kronsjö and Shumsheruddin provide a good summary of the four categories of parallel evolutionary algorithm implementations [52]. These four categories are as follows.

1. Synchronous master-slave
2. Semi-synchronous master-slave
3. Distributed, asynchronous concurrent
4. Network

In the first category, there is a master node that controls the selection and mating of individuals while the evaluation, typically the computational bottleneck, is performed by the slave processors. Since the computation time involved in evaluation is highly variable, slave processors can have large idle times waiting for the next individual to evaluate. This leads to the second category which overcomes the problem of synchronous tasking. The third category eliminates the master node altogether by storing population information in a common shared memory location and allowing each processor to perform selection, mating, and evaluation independently. The last category calls for a number of independent evolutionary algorithms to run in parallel occasionally broadcasting the best individuals to all of the other populations.

When there is a master-slave relationship, the challenge is to elegantly manage the communication overhead which increases with the number of slave processors used [26]. A distributed environment with shared memory for the population information is a similar situation, but access to the common memory becomes the bottleneck as the number of processors grows. With a network situation, no global information is available to the individual processors so the issue becomes how to partition the population and establish local selection rules [31].

Efforts in the network category usually fall into the category of island models [89] or neighborhood models [16, 60]. Island models are the more coarse-grained network variant. In island models, several separate evolutionary algorithms are run concurrently and fit individuals are exchanged periodically. Neighborhood models are the fine-grained network variant. In neighborhood models, the individuals of a population are distributed among the processors of a large mesh and rules are established for localized selection.

Kohlmorgen, Schmeck, and Haase have applied parallel evolutionary algorithm techniques to the RCPSP [46]. These authors implemented both island and neighborhood models on the massively parallel computer MasPar MP1 with 16k processing elements. They used this implementation to solve a wide variety of problems including the traveling salesman problem, the resource constrained

project scheduling problem, the uncapacitated warehouse location problem, the flow shop problem, and the capacitated lot-sizing problem. Their results with the RCPSP were particularly promising due to the incorporation of heuristic methods in the evaluation process.

*2.5.2 Parallel Tree Search Algorithms.* There are a number of different approaches to implementing branch and bound algorithms in parallel. Kronsjö and Shumsheruddin categorize the different methods as follows [52].

1. Parallel expansion of the search tree with different initial bound values
2. Parallel search using different algorithms
3. Perform the expansion of a single node in parallel
4. Parallel evaluation of subproblems

In a minimization problem, a branch and bound algorithm could obtain an initial upper bound from the best known feasible solution and an initial lower bound from a relaxation of the problem. The first method of parallelization would start several branch and bound instances on different processors dividing the range between the actual upper and lower bounds between them. Whenever a processor finds a feasible solution, a new upper bound is established and any processor whose work becomes bounded starts over with bounds in the new range. If a processor finishes without finding a feasible solution, then a new lower bound is established and the processor starts over with bounds in the new range. Again, any processor whose work becomes bounded must start over with new bounds. The process terminates when one processor completes the search with an optimal solution.

The second category of parallel branch and bound methods runs the same problem simultaneously on different processors with different branching strategies. For example, one processor could run a depth first search while another uses a best first search. Typically, it is not known which strategy works best for a given problem. By running different strategies in parallel, the benefits of both are combined and there is also the possibility of sharing the bounds obtained when one processor finds a new feasible solution.

The third method is particularly applicable when the evaluation of each node takes large computational effort. For example, in integer programming, an LP relaxation of the original problem

must be solved at every node. It would be ideal to implement a parallel LP solution algorithm to speed up the evaluation of nodes. This type of implementation would be highly problem specific.

The final, and most common, method of parallel branch and bound simply divides up the work of the sequential branch and bound algorithm by using multiple processors to evaluate different nodes simultaneously. The two most critical factors in an algorithm that takes this approach are how to divide the work between processors and how to communicate the new feasible solutions of one processor to all of the other processors. A new feasible solution represents a possible new bound for the other processors that could reduce their search space, but excessive communication overhead could slow the algorithm down in the long run.

Optimal parallel runtime is normally defined as  $T_p^* = T_s/p$  and this occurs when there is linear speedup. With this in mind, the bounds on expected parallel runtime are expressed in Equation (53).

$$\frac{T_s}{p} \leq T_p \leq T_s \quad (53)$$

$$T_p > T_s \quad (54)$$

$$T_p < \frac{T_s}{p} \quad (55)$$

Equations (54) and (55) represent the anomalies that can occur in parallel branch and bound implementations. The anomaly illustrated in Equation (54) is called a detrimental anomaly, while the one in Equation (55) is called an acceleration anomaly [42]. An acceleration anomaly leads to superlinear speedup and is possible because of the irregular nature of the problem. Detrimental anomalies are a cause for concern when implementing parallel branch and bound methods. Numerous efforts in the literature have focused on how to define conditions such that this sort of anomaly is guaranteed not to occur [15, 54, 59].

## 2.6 Testing Project Scheduling Algorithms

In order to gauge the value of an algorithm for solving project scheduling problems, it is necessary to obtain a set of sample problems to which the algorithm may be applied. Hall offers some guidelines for generating a set of project scheduling problems that is unbiased and representative of

the full range of possible problem types [33]. Jackson et al., while discussing guidelines on reporting the results of computational experiments, state the following common deficiencies in sample problem test sets [43].

1. The test set is usually small compared to the range of potential problems
2. The problems are often small and regular
3. The problems often have similar properties
4. Tuning an algorithm for a set of test problems may not give ideal performance in other settings
5. Inferring performance from a test set is similar to inferring the average height of Americans by sampling heights of the presidents

Jackson et al. recommend that standard test problem sets or test problem generators be used whenever possible [43]. In this way, the results from one algorithm may be compared against the results from other algorithms designed for the same problem and it is easier to support claims of relative efficiency. The Patterson test problems are the most widely used test set for the RCPSP, but they do not include problems where activities have multiple execution modes [70]. The only set of project scheduling problems for the MMRCPPSP is contained in PSPLIB, a set of benchmark instances for resource constrained project scheduling [48, 49].

There are also a few publicly available tools for randomly generating test problems. Hall [33] and Demeulemeester, Dodin, and Herroelen [22] have developed network generation software to generate network structures of a specified density to form the precedence constraints of a project scheduling problem. Kolisch, Sprecher, and Drexel have developed ProGen, a problem generator that can generate MMRCPPSP problem instances of a controlled difficulty [50, 51].

## *2.7 Summary*

This chapter reviewed the literature that provides a background for the research in this dissertation. The review focused on the broad area of project scheduling, including model formulations, solution methodologies, and approaches to testing. In Chapter III, a project scheduling model is formulated for the force level air campaign planning problem. Additionally, Chapter III outlines the solution methodologies that this research develops for the air campaign planning problem.



### *III. Methodology*

The purpose of this chapter is to provide an overview of the methodology used to accomplish the research objectives outlined in Chapter I. The overall research goal is to develop an approach for solving large combat planning problems using integer program decomposition. In order to achieve this goal, a new variant of the resource constrained project scheduling problem (RCPSP) is formulated, several algorithms are developed to solve this new problem, and a case study is accomplished to show the applicability of the approach to joint campaign planning, specifically to the force level planning of an air campaign.

The chapter is divided into two sections. The first section presents the development of an original RCPSP formulation which is suitable for modeling the problem of optimally planning combat missions in an air campaign. The second section provides an overview of the algorithms this research develops specifically for solving large instances of this combat planning problem.

#### *3.1 Combat Planning Model Formulation*

The problem modeled in this section is that of generating all the combat missions to be flown in an ATO planning period. Recall that generating combat missions for an ATO requires matching available air assets against nominated targets and then scheduling the resulting combat sorties. At first glance, it would seem that this process could be modeled as an assignment problem; however, the assets are not continuously available and it is necessary to enforce certain timing requirements among the missions.

The requirement for mission scheduling, mission precedence, and resource accounting indicates that a RCPSP formulation is in order. A review of the literature reveals that none of the available RCPSP formulations provide for all of the aspects of combat planning that are necessary in an air campaign. It is necessary to combine features from several RCPSP variants to develop an appropriate model. This new model formulation includes multi-modal activities, doubly constrained resources, and generalized precedence constraints. The model is referred to as the multi-modal generalized resource constrained project scheduling problem (MMGRCPSP).

*3.1.1 Multi-modal Activities.* The first aspect of the MMGRCPSP to consider is the multi-modal nature of the activities. The activities that occur in an air campaign are the strikes by friendly air assets against enemy targets. The first component of any strike is the target. The target nomination list (TNL) may be used to determine the activities in an air campaign planning problem. Early in the ATO generation cycle, a set of weaponeering options is compiled for every target on the TNL. These options are determined based on the availability of aircraft, crews, and munitions as well as a threshold value indicating the acceptable level for probability of kill (PK). Table 1 provides a sample TNL entry.

Table 1 A Sample TNL Entry							
Target	Priority	Window	Option	Sorties	Aircraft	SCL	PK
101	1	0800/0900	1	2	F111	A1	85%
			2	4	F16	B3	95%

Table 1 gives an example of the type of information that a TNL normally provides to combat planners. The table contains a TNL entry for target number 101, a priority 1 target with a window of opportunity from 0800 until 0900. Two weaponeering options have been deemed acceptable for this target. One option calls for the target to be attacked by a mission of two F111s using standard configuration load (SCL) A1. The expected PK for this option is 85%. The other option requires four F16 sorties with SCL B3 and has an expected PK of 95%.

The sample TNL entry in Table 1 indicates that there are two acceptable weaponeering options for target 101, but this does not necessarily correspond to an activity with two possible execution modes. Suppose that for a given air campaign planning scenario F16s are located at Base A while F111s are located at both Base B and Base C. The activity that is used to model target 101 for this scenario has the following three possible execution modes.

- Mode 1 4 F16s from Base A
- Mode 2 2 F111s from Base B
- Mode 3 2 F111s from Base C

From this example it is clear that the acceptable execution modes for every target, along with the resources required by each mode, may be taken directly from the TNL. The final piece of information necessary to completely describe activities in the MMGRCPSP formulation for air

campaign planning is activity duration. The duration of an activity depends on the location of the target modeled by that activity and the type and location of the resources required by the execution mode selected for that activity. A duration may be determined for each mode of every activity using the location of the assets, the location of the resources, and nominal air speed values to perform rudimentary routing calculations.

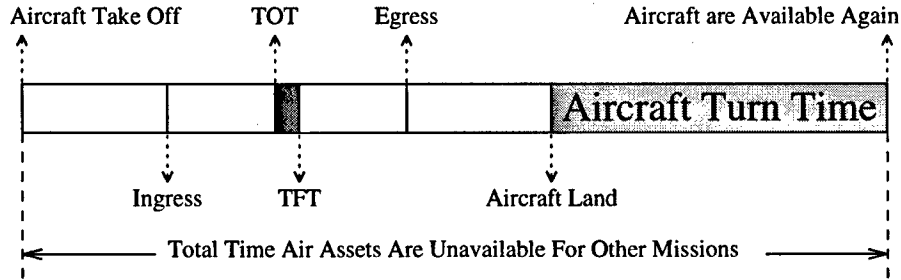


Figure 5 The Components of Mission Duration

Figure 5 illustrates the the various mission legs that must be considered when calculating the duration for each mode of an activity. It is important for resource accounting reasons that activity durations include the entire mission time from the time the aircraft take off until the time they have landed, been serviced, and are ready for another mission. In Figure 5, take off and landing are self explanatory, ingress and egress mark the points in time when the aircraft transition between friendly and enemy territories, and time on target (TOT) and time off target (TFT) denote the time window available for the mission to attack its target. It is important to note that the time window indicated by TOT/TFT represents a window of availability, not a loiter time. It is possible that in some operational scenarios, more components may be required to accurately model some combat missions, but in this dissertation, without loss of generality, all missions are modeled with the same six duration components illustrated in Figure 5.

To model multi-modal activities, the MMGRCPSP formulation for air campaign planning takes its variables, objective function, and one of its constraints directly from the MMRCPPSP formulation in Chapter II. Every activity  $i$  has a set of binary variables,  $x_{imt} \in \{0, 1\}$ , that specify the execution mode and start time for that activity. If  $x_{imt} = 1$ , activity  $i$  begins execution in mode  $m$  at time  $t$ . Since an activity can only have one start time and one mode of execution, the set of binary variables for each activity forms a special ordered set. Exactly one variable in each special

ordered set must be assigned a value of 1. The formulation of the constraints that enforce this condition is given by Equation (56).

$$\sum_{m \in M_i} \sum_{t=e_{im}}^{l_{im}} x_{imt} = 1 \quad \forall i \in A \quad (56)$$

In Equation (56),  $A$  is the set of all activities.  $M_i$  is the set of every possible execution mode for activity  $i \in A$ . The possible start times for activity  $i$  in mode  $m$  include every integer value from the earliest possible start time,  $e_{im}$ , to the latest possible start time,  $l_{im}$ . The simplest method of assigning values for the earliest and latest possible start times is to select values that cover the entire ATO planning period. However, this method leads to a model formulation with a large number of unnecessary binary variables. CPM methods may be applied to eliminate infeasible start time/execution mode combinations in order to reduce the range of  $[e_{im}, l_{im}]$  and thereby reduce the number of variables in the model.

The objective of the MMGRCPSP is to minimize makespan, which has the effect of minimizing the total time to complete all missions. This is accomplished by minimizing the completion time of the problem's terminal activity, activity  $d$ . The duration of activity  $d$  in mode  $m$  is given by the parameter  $\tau_{dm}$ . If the activity  $d$  begins execution in mode  $m$  at time  $t$ , the completion time of the terminal activity, and the objective function value of the problem, is given as  $t + \tau_{dm}$ . The complete objective function formulation is provided by Equation (57).

$$\text{Minimize} \quad \sum_{m \in M_d} \sum_{t=e_{dm}}^{l_{dm}} (t + \tau_{dm}) x_{dmt} \quad (57)$$

**3.1.2 Doubly Constrained Resources.** This section presents the framework that is used to model air assets as resources in the MMGRCPSP formulation for air campaign planning. The previous discussion on determining activity execution modes from a target nomination list made the point that the MMGRCPSP formulation must model each aircraft type/location combination with a separate project resource. For example, the F111s at Base B are modeled with a separate project resource than the resource that is used to represent the F111s at Base C.

As explained below, each of the resources used to represent these F111 units is both a renewable resource and a nonrenewable resource, a condition that is referred to as *doubly constrained*.

Every unit in an air campaign planning scenario is represented as a doubly constrained resource in the MMGRCPSP formulation.  $K$  is the set of all doubly constrained resources (units) in the problem. The renewable aspect of each resource  $k \in K$  is the limit dictated by  $R_k$ , the actual number of aircraft that make up the unit represented by resource  $k$ . At any given time during the ATO planning period, no more than  $R_k$  units of resource  $k \in K$  may be in use. The formulation of the constraints that enforce this condition is given by Equation (58). In this constraint formulation,  $g$  represents the project deadline (the end of the ATO day) and  $r_{imk}$  is the number of units of resource  $k$  required by activity  $i$  when executed in mode  $m$ .

$$\sum_{i \in A} \sum_{m \in M_i} \sum_{t=j-\tau_{im}+1}^j r_{imk} x_{imt} \leq R_k \quad \forall k \in K \text{ and } j = 1, \dots, g \quad (58)$$

The nonrenewable aspect of each resource  $k \in K$  is the limit dictated by  $N_k$ , the number of sorties that unit  $k$  can generate in a given ATO planning period. Generally, each aircraft in a unit can fly more than one mission in a given ATO day as long as there are crews, fuel, and munitions available. Each unit is assigned a turn rate value  $tr_k$  and the number of sorties available from that unit is given as  $N_k = R_k \times tr_k$ . This number is truncated to the nearest integer. While  $R_k$  represents the maximum number of units of resources  $k$  that may be in use at any given time,  $N_k$  provides the upper bound on the total number of units of resource  $k$  available. The formulation of the constraints that enforce the nonrenewable resource condition is given by Equation (59).

$$\sum_{i \in A} \sum_{m \in M_i} \sum_{t=e_{im}}^{l_{im}} r_{imk} x_{imt} \leq N_k \quad \forall k \in K \quad (59)$$

To illustrate how the constraints given by Equations (58) and (59) enforce the limitations on both the number of aircraft in a unit and the number of sorties that unit can provide, consider a unit of 16 F16s that has a turn rate of three. This turn rate indicates that each aircraft can fly at most three missions in a given ATO day. The renewable resource constraints limit the number of sorties available from that unit at any given time to 16 or less. The nonrenewable resource constraints limit the total number of sorties flown by that unit to 48 or less. Figure 6 shows a sortie utilization for this unit that is feasible with respect to both the renewable and the nonrenewable resource constraints.

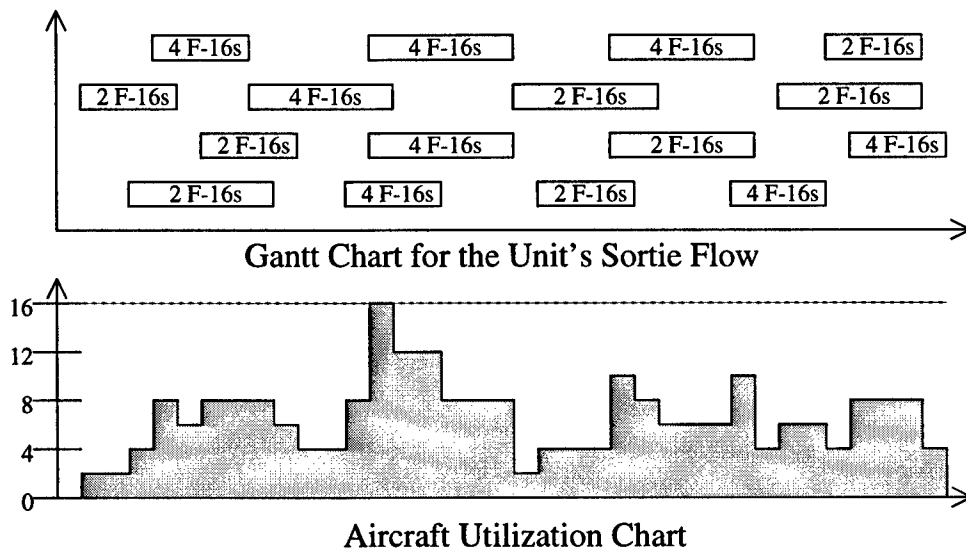


Figure 6 A Feasible Utilization of Sorties

The horizontal axis for both graphs in Figure 6 represents time. Each of the 16 rectangles on the Gantt chart depicts the entire duration of a combat mission flown by the F16 unit in the example. Of the 16 missions on the Gantt chart, eight require four sorties from the unit and the other eight require only two sorties. This indicates a requirement for a total of 48 sorties which is feasible according to the nonrenewable resource constraint described previously.

The second chart in Figure 6 translates the sortie utilization illustrated by the Gantt chart into a function of aircraft utilization over time. The dotted line on the aircraft utilization chart represents the number of aircraft that make up the unit. Since the function of aircraft utilization over time never exceeds the threshold value given by the dotted line, the sortie utilization shown is feasible with respect to the renewable resource constraints described previously.

The charts in Figure 6 provide more than just feasibility information. While the solution corresponding to this utilization of resources may be feasible, it also has some less favorable qualities. The utilization of both aircraft and sorties is at the upper bound indicating that the loss of a single aircraft makes the solution infeasible. This resource utilization information might prove useful when a commander can choose between several near optimal solutions.

*3.1.3 Generalized Precedence Constraints.* The final component of the MMGRCPSP formulation for air campaign planning is mission precedence. The variables and constraints formulated so far provide for activities and resources. All that remains is to specify the constraints to enforce the timing between various activities. Standard end to start activity precedence relations are not flexible enough to enforce the mission sequencing necessary in air campaign planning. It is necessary to use generalized precedence constraints.

Standard end to start activity precedence specifies that if activity  $i$  is a predecessor of activity  $j$ , activity  $j$  cannot begin execution until the execution of activity  $i$  has terminated. This type of precedence applied to combat missions would only be able to enforce that mission  $j$  may not take off until all aircraft involved in the execution of mission  $i$  have landed and been serviced. This is not the precision strike timing for which our Air Force is noted.

Generalized precedence constraints, on the other hand, may be used to enforce any timing requirement called for in an operational scenario. An example of the type of mission sequencing that may be modeled using generalized precedence constraints is provided by Figure 7. The generic map in Figure 7 depicts friendly airspace and enemy airspace divided by a line denoting the forward line own troops (FLOT). On the friendly side there are two bases and on the enemy side there are two targets.

Suppose the target nearer the FLOT is an enemy air defense target and the other is a deep interdiction target, perhaps a headquarters building. Note that the dashed line that represents the path of the interdiction mission passes close to the enemy air defense site. It might be important to a combat planner that the mission to destroy the enemy air defense target, the dotted line on the map, hits its target before the interdiction mission enters enemy airspace. This type of mission timing may be enforced by applying generalized precedence constraints to dictate a minimum lag time between the start of the enemy air defense mission and the start of the interdiction mission. This minimum lag time value depends on the execution modes of both the predecessor and the successor missions.

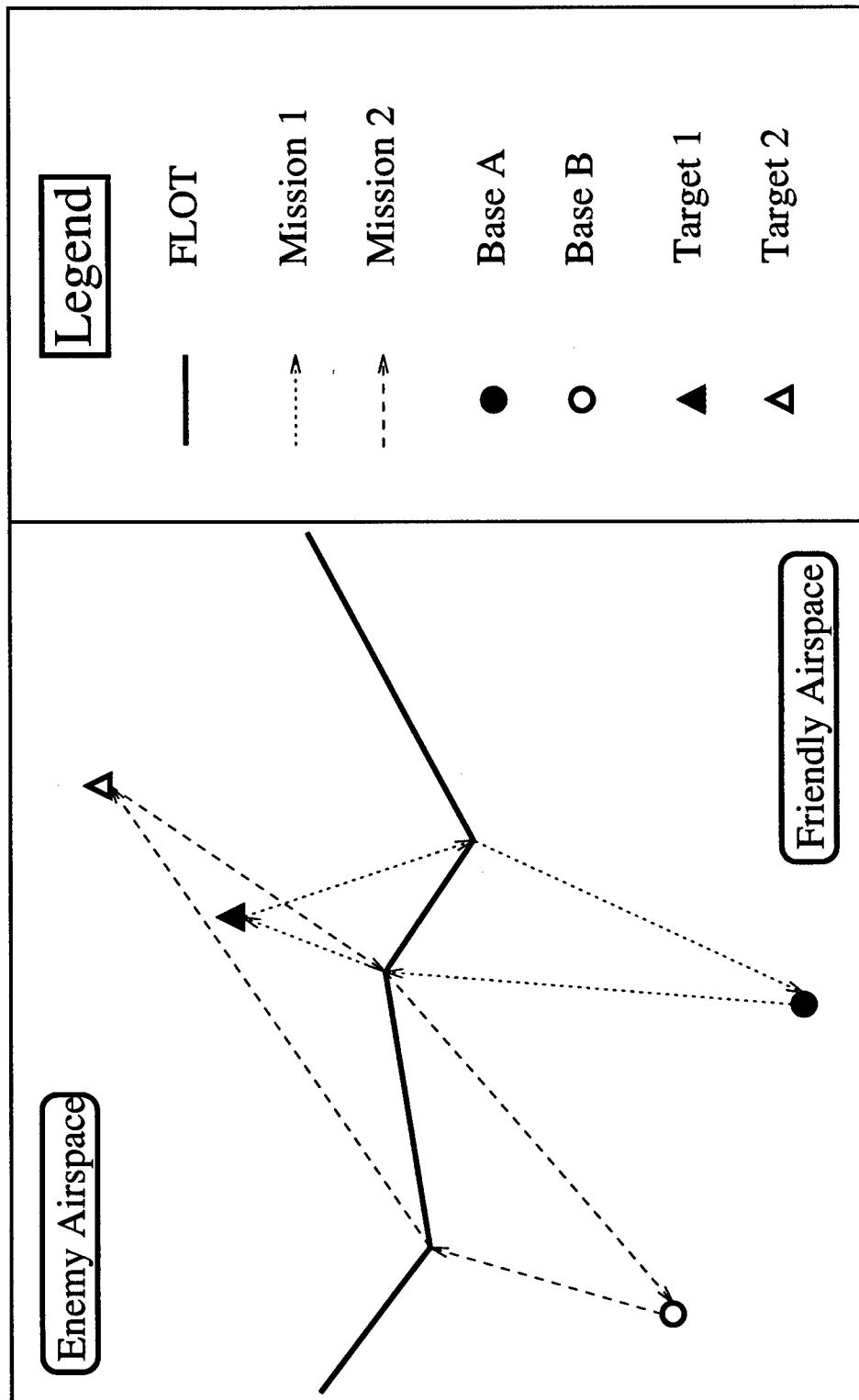


Figure 7 A Scenario for Generalized Precedence



Figure 8 demonstrates how to determine the value for the minimum lag between the start of the enemy air defense mission  $i$  in mode  $m$  and the start of the interdiction mission  $j$  in mode  $n$ . The figure illustrates that the ingress of mission  $j$ , the successor mission, must occur after the attack window of mission  $i$ , the predecessor mission. Let  $ST_i$  be the start time of mission  $i$ ,  $ST_j$  be the start time of mission  $j$ ,  $\delta_i$  be the time between the take off of mission  $i$  and the end of its attack window, and  $\delta_j$  be the time between the take off of mission  $j$  and its ingress. The minimum lag value is given by  $\Delta_{ijmn} = \delta_i - \delta_j$  and the generalized precedence constraint is summarized by  $ST_j - ST_i \geq \Delta_{ijmn}$ .

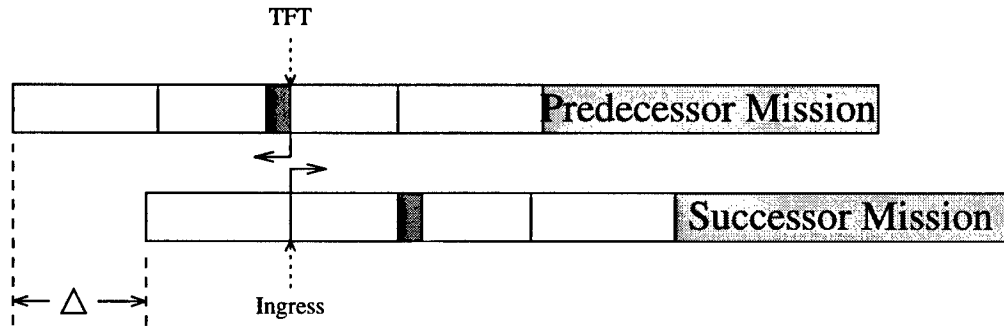


Figure 8 Determining the Minimum Lag Value

It must be stressed that since the components of mission duration depend on the location of both assets and targets, as well as the nominal air speed of the aircraft involved, there is a different minimum lag value associated with every possible combination of execution modes  $(m, n)$ . If there are three possible execution modes for activity  $i$  and four possible execution modes for activity  $j$ , there are a total of 12 possible minimum lag values that correspond to 12 different formulations for the constraint that enforces precedence between activities  $i$  and  $j$ . All but one of these precedence constraints must be relaxed depending on the choice of modes for activities  $i$  and  $j$ .

Standard integer program formulation techniques might be applied to formulate all 12 possible precedence constraints and use additional binary variables to selectively enforce or relax each constraint depending on the execution modes of activities  $i$  and  $j$ . While this approach would achieve the goal of modeling generalized precedence constraints, it would require the addition of a large number of variables and constraints to a model that is already unwieldy. The constraint formulation given by Equation (60) allows all 12 possible precedence formulations to be included using

three constraints and the existing model variables to implement the logic that selectively enforces or relaxes the various precedence conditions. In this constraint formulation,  $S_i$  is the set of all general successors to activity  $i$ .

$$\sum_{n \in M_j} \sum_{t=e_{jn}}^{l_{jn}} (\Delta_{ijm'n} - t)x_{jnt} + \sum_{t=e_{im'}}^{l_{im'}} tx_{im't} \leq \sum_{m \in (M_i \setminus m')} \sum_{t=e_{im}}^{l_{im}} gx_{imt} \quad \forall i \in A, j \in S_i, m' \in M_i \quad (60)$$

To clarify the logic enforced by the constraint set given by Equation (60), consider the constraint corresponding to some predecessor activity  $i$ , its general successor activity  $j \in S_i$ , and one of its possible execution modes  $m' \in M_i$ . Let  $m$  be the execution mode selected for activity  $i$  and let  $n$  be the execution mode selected for activity  $j$ . The precedence constraint associated with  $i, j$ , and  $m'$  can be reduced to one of two possible forms depending on the relationship of  $m'$  to  $m$ .

- IF  $m' = m$  THEN  $\Delta_{ijm'n} - ST_j + ST_i \leq 0 \quad \rightarrow \quad ST_j - ST_i \geq \Delta_{ijmn}$
- IF  $m' \neq m$  THEN  $\Delta_{ijm'n} - ST_j + 0 \leq g \quad \rightarrow \quad ST_j \geq \Delta_{ijm'n} - g$

The constraint is only important if  $m' = m$ . If  $m' \neq m$ , the constraint becomes redundant since  $\Delta_{ijm'n} - g$  is a large negative number and all start times are already constrained to be positive. When  $m' = m$ , the constraint enforces the appropriate minimum lag between the start times of activities  $i$  and  $j$  in modes  $m$  and  $n$ .

This formulation of precedence constraints is significant for two reasons. First, a review of the literature shows that none of the available RCPSP models incorporate both multi-modal activities and generalized precedence constraints. Second, the formulation is significant for its unique method of enforcing precedence condition selection logic with a minimal number of constraints and no additional variables.

**3.1.4 Complete Model Formulation.** The complete MMGRCPSP model formulation for air campaign planning is summarized by Table 2 and Figure 9.

Table 2 MMGRCPSP Variables and Parameters

$A$	the set of all activities
$d$	the index of the terminal activity
$e_{im}$	the earliest start for activity $i$ in mode $m$
$l_{im}$	the latest start for activity $i$ in mode $m$
$M_i$	the set of all execution modes for activity $i$
$\tau_{im}$	the duration of activity $i$ in mode $m$
$S_i$	the set of generalized successors of activity $i$
$\Delta_{ijmn}$	the minimum lag between the start time of activity $i$ in mode $m$ and the start time of activity $j \in S_i$ in mode $n$
$K$	the set of all doubly constrained resources
$r_{imk}$	the number of units of resource $k$ required by activity $i$ when being executed in mode $m$
$R_k$	the per period availability of resource $k$
$N_k$	the total amount of resource $k$ available
$g$	the deadline for the project under consideration
$x_{imt}$	1 if activity $i$ starts in period $t$ using mode $m$

---


$$\begin{aligned}
 &\text{Minimize} \quad \sum_{m \in M_d} \sum_{t=e_{dm}}^{l_{dm}} (t + \tau_{dm}) x_{dmt} \\
 &\text{Subject to} \quad \sum_{i \in A} \sum_{m \in M_i} \sum_{t=e_{im}}^{l_{im}} r_{imk} x_{imt} \leq N_k \quad \forall k \in K \\
 &\quad \sum_{i \in A} \sum_{m \in M_i} \sum_{t=j-\tau_{im}+1}^j r_{imk} x_{imt} \leq R_k \quad \forall k \in K, j = 1, \dots, g \\
 &\quad \sum_{n \in M_j} \sum_{t=e_{jn}}^{l_{jn}} (\Delta_{ijm'n} - t) x_{jnt} + \sum_{t=e_{im'}}^{l_{im'}} t x_{im't} \leq \sum_{m \in (M_i \setminus m')} \sum_{t=e_{im}}^{l_{im}} g x_{imt} \quad \forall i \in A, j \in S_i, m' \in M_i \\
 &\quad \sum_{m \in M_i} \sum_{t=e_{im}}^{l_{im}} x_{imt} = 1 \quad \forall i \in A \\
 &\quad x_{imt} \in \{0, 1\} \quad \forall (i, m, t)
 \end{aligned}$$


---

Figure 9 The Complete MMGRCPSP Model Formulation

### 3.2 *Combat Planning Solution Methods*

The research in this dissertation is presented in blocks such that the first block develops a solution methodology for the MMGRCPSP. Each additional block offers extensions that build on the previous blocks in order to exploit the structure of large problem instances to reduce solution times. Each major block of research is provided in a separate chapter with one additional chapter to present a case study that demonstrates the applicability of the solution methodologies to a large combat planning problem.

The overall goal of the research is to develop an effective integer program decomposition approach to combat planning, but before a decomposition algorithm may be implemented, it is necessary to develop a subproblem solution methodology. Therefore, the first block of research is devoted to developing an algorithm for solving small to medium sized MMGRCPSP instances. The next block of research develops an integer program decomposition approach for solving large block-angular MMGRCPSP instances. The last block of research develops a heuristic approach to the problem using evolutionary algorithm techniques and then combines the evolutionary algorithm with the decomposition algorithm in a hybrid approach.

*3.2.1 Subproblem Solution Methodology.* The most efficient exact solution procedures found in the literature review for resource constrained project scheduling problems were the implicit enumeration algorithms presented by Demeulemeester [21, 23, 24, 25] and Sprecher [79, 80, 81]. Of the procedures found in these works, Sprecher's procedure for the MMRCPSPP is the closest to what is required for the subproblems developed in this research. Sprecher's procedure can handle multiple activity execution modes, doubly constrained resources, and minimum makespan objective functions.

However, one important feature not included in Sprecher's procedure is the capability to address the generalized precedence requirements required to accurately model combat mission timing in an air campaign planning problem. Another important consideration for the subproblem solution methodology is the type of decomposition to be used. The Sweeney-Murphy decomposition algorithm requires that  $k$ -best solution sets be generated for the subproblems; another important feature that Sprecher's procedure does not provide.

While Sprecher's algorithm may not be applied directly to the subproblems of the decomposition effort in this dissertation, it serves as a starting point for developing a specialized algorithm for the MMGRCPSP. The first step in developing the subproblem solution algorithm is to prove that Sprecher's implicit enumeration approach, extended to allow for generalized precedence and  $k$ -best solution sets, is guaranteed to find an optimal solution for every feasible MMGRCPSP instance. The mechanics of the extended algorithm are then defined and the solution methodology is implemented in C to allow performance of empirical tests.

*3.2.2 Decomposition Methodology.* The decomposition methodology selected for use in this dissertation is an integer program decomposition technique developed by Sweeney and Murphy [84]. Sweeney-Murphy decomposition is the only general decomposition approach available in the literature for block-angular integer programs. While Sweeney-Murphy decomposition is a theoretically sound technique, there are limited applications of the approach in the literature.

There are some potentially cumbersome aspects of the Sweeney-Murphy approach. The first of these cumbersome aspects is the requirement to generate  $k$ -best subproblem solution sets. The literature offers no generalized procedure for obtaining  $k$ -best solution sets to integer problems other than iteratively solving the problem and then cutting the optimal solution out of the feasible region. This *brute force* approach to the subproblems may require so much computational effort that the decomposition approach loses its appeal.

Another open question in using Sweeney-Murphy decomposition is the requirement for Lagrangian multipliers to formulate the subproblems. The techniques that Sweeney and Murphy offer for calculating multipliers are not feasible for all types of integer problems. For certain classes of problems, including the MMGRCPSP, it is difficult to obtain appropriate multipliers and inappropriate multiplier values may lead to excessive computation time.

The final open issue of Sweeney-Murphy decomposition involves setting and resetting the values of  $k_i$  that control the progress of the overall algorithm. The  $k$  selection strategy used in an implementation of the Sweeney-Murphy approach has a large effect on the computational efficiency of the algorithm. However, in every instance where Sweeney-Murphy decomposition is used in the literature, the  $k$  selection strategy is arbitrary with no justification given for the values used.

The challenge of the decomposition research in this dissertation is to develop a solution methodology that overcomes the cumbersome aspects of Sweeney-Murphy decomposition and offers an effective approach that exploits block-angular structure in large MMGRCPSP instances to reduce solution time. The resulting decomposition algorithm is implemented in C to allow empirical testing.

*3.2.3 Heuristic Methodology.* Given that there are always limitations imposed by the time and hardware available, it is not always feasible to use exact solution methodologies on every problem instance. For this reason, the last block of research in this dissertation develops a heuristic approach to the MMGRCPSP. This approach applies evolutionary algorithm techniques to provide a heuristic solution methodology for problems that for reasons of size or lack of structure are not suitable for the exact methods developed in this dissertation.

There are several evolutionary algorithm approaches to resource constrained project scheduling available in the literature. However, none of these approaches offer the combination of multimodal activities and generalized precedence constraints that is necessary for the air campaign planning model developed in this dissertation. Fortunately, Hartmann provides an evolutionary algorithm for the MMRCPSPP formulation [35].

Since Hartmann's evolutionary algorithm stochastically searches the same solution space that is implicitly enumerated by Sprecher's algorithm, no further work is necessary to show that Hartmann's approach may be extended to include generalized precedence. The proof offered for the subproblem solution algorithm applies directly to the evolutionary algorithm efforts. Once the mechanics of the extended evolutionary algorithm have been defined, the approach is implemented in C, allowing empirical tests to be performed.

Beyond its value as a heuristic alternative to exact solution methodologies, the evolutionary algorithm developed in this dissertation provides a means of developing a hybrid decomposition approach that may reduce the solution time required for large problem instances. The standard decomposition approach developed in this dissertation requires repeated solution of the subproblems in order to use parametric analysis to estimate Lagrangian multiplier values. The same parametric analysis may be performed using the evolutionary algorithm to obtain the same type of estimates

at a much smaller cost in terms of computation time. The hybrid decomposition approach is implemented in C, allowing empirical tests to be performed.

*3.2.4 Test and Evaluation.* The testing performed in this dissertation is accomplished in three phases. In the first phase, the basic subproblem solution algorithm is applied to a wide range of test problems in order to evaluate its performance on various types of problems. As each additional block of research is presented, further testing is conducted to estimate the reduction in solution time offered by each extension to the basic subproblem solution methodology. The final phase of testing consists of a case study that demonstrates the applicability of the solution methodologies to a large combat planning problem.

In all but the final phase of testing, the test problems are generated using ProGen, a problem generator for project scheduling problems developed by Kolisch, Sprecher, and Drexel [50, 51]. This program, presented in 1995, has become the standard for generating project scheduling test problems. Three parameters that may be used to characterize resource constrained project scheduling problem instances are resource demand, resource availability, and precedence constraint network density. ProGen provides the ability to automatically generate test problems that exhibit whatever levels of these parameters are of interest. With this ability, ProGen may be used to generate sets of test problems that span the possible levels of these parameters in order to determine what parameter levels lead to the most difficult problem instances [12]. However, ProGen does not generate problems with generalized precedence relations. It is necessary to transform the precedence relations in the test problems provided by ProGen in order to obtain sample MMGRCPSP instances.

The final phase of testing involves a case study. The case study discusses how to formulate a MMGRCPSP from the data available in an actual air campaign planning scenario. However, the problem that is investigated in the case study is a notional example. The example problem is generated in order to demonstrate several different approaches to problem decomposition and several different applications of generalized precedence constraints. The hybrid decomposition approach is applied to the example problem to obtain the optimal solution along with a number of near optimal solution alternatives and several criteria are presented to discriminate among near optimal solutions on the basis of resources utilization.

### *3.3 Summary*

This chapter presented an overview of the methodology of this dissertation. The air campaign planning model formulation was described in detail and the solution methodologies developed for this model were summarized. Chapter IV presents the results of the first block of research, developing a subproblem solution algorithm. The algorithm from the literature that serves as the basis for the subproblem solution algorithm is described, an implicit enumeration approach for the MMGRCPSP is developed, and empirical results are obtained by applying the approach to representative sets of test problems.



#### *IV. Subproblem Solution Algorithm*

Any attempt to apply decomposition techniques to MMGRCPSP instances requires a method for solving the subproblems. These subproblems are merely smaller instances of the original MMGRCPSP being decomposed. With this in mind, the subproblem solution algorithm must be able to solve small to moderate size MMGRCPSP instances as rapidly as possible. The subproblem solution algorithm must also be able to generate the top  $k$  solutions for each subproblem in order to form the  $k$ -best solution sets required for the Sweeney-Murphy integer program decomposition method.

This chapter presents an algorithm for solving the subproblems of a decomposed MMGRCPSP instance. The algorithm is an implicit enumeration approach based on a similar approach proposed by Sprecher [79]. To solve the MMGRCPSP subproblems, Sprecher's algorithm is extended to deal with problems having generalized precedence constraints. Sprecher's algorithm is also extended to generate the  $k$ -best solution sets required by the Sweeney-Murphy algorithm. The resulting subproblem solution algorithm is applied to test problems to investigate the relationship between problem solution time and the value of certain parameters that govern the complexity of MMGRCPSP instances.

##### *4.1 Extending Sprecher*

The RCPSP has received a great deal of attention in the literature. Many algorithmic and heuristic approaches have been presented for the numerous variants of this general problem. Recent research in this area has focused on the multi-modal formulation of the problem, the MMRCPPSP. This effort looks at a variant that includes activities with both multiple execution modes and generalized precedence constraints, the MMGRCPSP. Specialized algorithms have been developed to handle activities with multiple execution modes and standard end to start precedence relations, but a review of the literature has not revealed research addressing multi-modal activities with generalized precedence constraints [13].

A transformation may be applied to convert the precedence constraints in any project scheduling problem from the generalized form to the standard form by breaking each of the original project activities up into several smaller component activities. Unfortunately, the resulting problem formulation of such a division of activities requires more constraints and variables than the original

formulation. If the additions are not too numerous, existing specialized algorithms may be applied to such a larger formulation. While specialized algorithms may solve this expanded problem formulation faster than a standard integer program solver could solve the original, smaller problem formulation, the increase in variables and constraints limits the scale of problems that may be approached. The goal of this section is to develop a specialized algorithm for the MMGRCPSP itself rather than reformulating it as a larger MMRCPPSP and using existing algorithms.

A review of the literature shows that although there is no specialized algorithm available for the MMGRCPSP, there are many algorithms available for other variants of the RCPSP. Of the algorithms available, Sprecher's algorithm addresses more aspects of the MMGRCPSP than any of the others, and therefore provides a good starting point for developing a subproblem solution algorithm for the proposed decomposition approach. Sprecher's algorithm is an implicit enumeration approach and is currently the fastest exact algorithm available in the published literature for the MMRCPPSP [72, 80]. The following sections address how Sprecher's algorithm is extended to include  $k$ -best solution sets, generalized precedence constraints, and regular performance measures other than makespan.

**4.1.1 Convergence.** Sprecher, Kolisch, and Drexel review several standard job shop problem definitions, then extend and formalize them for the RCPSP [82]. These definitions, given below, are extended here to address the issue of multiple execution modes and are necessary for a discussion of the convergence of Sprecher's implicit enumeration algorithm.

**Definition 1** Consider a project with  $J$  multi-modal activities. Let  $ST_j$  and  $m_j$  represent the start time and execution mode of activity  $j$ , respectively. A schedule,  $S = (S, M)$ , is a combination of two  $J$ -tuples,  $S = (ST_1, \dots, ST_J)$  and  $M = (m_1, \dots, m_J)$ , where  $S$  and  $M$  provide the start time and execution mode for every project activity  $j$ , where  $j = 1, \dots, J$ .

**Definition 2** Consider a feasible schedule  $S = (S, M)$ . A left shift of activity  $j$ ,  $1 \leq j \leq J$ , is an operation on the  $J$ -tuple of start times,  $S$ , which derives some new feasible  $J$ -tuple of start times,  $S'$ , such that  $ST'_j < ST_j$  and  $ST'_i = ST_i$  for  $i$ , where  $i = 1, \dots, J$  and  $i \neq j$ .

**Definition 3** A left shift of activity  $j$ ,  $1 \leq j \leq J$ , is called a one-period left shift, if  $ST_j - ST'_j = 1$ .

**Definition 4** A local left shift of activity  $j$ ,  $1 \leq j \leq J$ , is a left shift of activity  $j$  which is obtainable by one or more successively applied one-period left shifts of activity  $j$ .

**Definition 5** A global left shift of activity  $j$ ,  $1 \leq j \leq J$ , is a left shift of activity  $j$  which is not obtainable by a local left shift.

**Definition 6** A semi-active schedule is a feasible schedule, where none of the activities  $j$ ,  $1 \leq j \leq J$ , can be locally left shifted.

**Definition 7** An active schedule is a feasible schedule, where none of the activities  $j$ ,  $1 \leq j \leq J$ , can be locally or globally left shifted.

RCPSP implicit enumeration algorithms like Sprecher's build solutions incrementally as they explore deeper into the solution tree. At each level  $i$  of the tree, some activity  $g_i$  is scheduled to begin execution in mode  $m_{g_i}$  at time  $ST_{g_i}$ . At all but the final stage of the algorithm, only a subset of the project's activities have been scheduled. The scheduling information for this subset of activities is called a partial schedule and is defined as follows:

**Definition 8** For some  $i < J$ , an  $i$ -partial schedule  $PS_i$  is an  $i$ -tuple of quadruples

$$PS_i = \begin{pmatrix} 1 & 2 & \dots & i \\ g_1 & g_2 & \dots & g_i \\ m_{g_1} & m_{g_2} & \dots & m_{g_i} \\ ST_{g_1} & ST_{g_2} & \dots & ST_{g_i} \end{pmatrix},$$

where  $(j, g_j, m_{g_j}, ST_{g_j})$  outlines that on level  $j$ , activity  $g_j$  is scheduled to begin execution in mode  $m_{g_j}$  at time  $ST_{g_j}$ .

**Definition 9** Let  $PS_i$  be an  $i$ -partial schedule. Now, consider the schedule  $S' = (S', \mathcal{M}')$ , where  $S' = (ST'_1, \dots, ST'_J)$  and  $\mathcal{M}' = (m'_1, \dots, m'_J)$ . Schedule  $S'$  is called a completion of the  $i$ -partial schedule  $PS_i$  if  $m'_{g_k} = m_{g_k}$  and  $ST'_{g_k} = ST_{g_k}$  for all  $k = 1, \dots, i$ .

Two additional definitions are required before the convergence of Sprecher's algorithm may be discussed. The first definition describes the type of solutions the algorithm implicitly enumerates

and the second defines the class of performance measures for which the algorithm is guaranteed to converge.

**Definition 10** Consider a permutation of all activities  $1, \dots, J$  denoted by the  $J$ -tuple  $\mathcal{G} = (g_1, \dots, g_J)$ . Let  $\mathbf{S} = (\mathcal{S}, \mathcal{M})$  be a feasible schedule where the activity start times are obtained by considering the activities individually in the sequence defined by  $\mathcal{G}$  and scheduled to begin execution in the mode defined by  $\mathcal{M}$  at the earliest possible time such that no precedence or resource constraint is violated and  $ST_{g_i} \leq ST_{g_{i+1}}$  for all  $i = 1, \dots, (J - 1)$ . The schedule  $\mathbf{S}$  is called an *active permutation schedule*.

**Definition 11** Consider a scheduling problem where the objective is to minimize  $\Phi$ , a measure of schedule fitness. Let  $\mathbf{S} = (\mathcal{S}, \mathcal{M})$  be a feasible schedule for the problem and let  $\Phi(\mathcal{S}, \mathcal{M})$  represent the fitness of schedule  $\mathbf{S}$ .  $\Phi$  is called a *regular measure of performance* if  $\Phi(\mathcal{S}, \mathcal{M}) < \Phi(\mathcal{S}', \mathcal{M})$  implies that  $ST_j < ST'_j$  for at least one  $j$ ,  $j \in \{1, \dots, J\}$ .

The variable and parameter definitions in Table 3, along with Equations (61) through (66) in Figure 10, give the complete formulation of the MMGRCPSP used in this dissertation. The significant difference between this formulation and Sprecher's MMRCPPSP formulation is the statement of generalized precedence relations given by Equation (64).

Sprecher's algorithm implicitly enumerates every feasible *active permutation schedule* for a given problem [79]. Sprecher shows that his implicit enumeration algorithm converges to optimality for any regular measure of performance by proving that if a problem is feasible, it must have an active permutation schedule that is optimal. His proof uses the properties of standard end to start activity precedence to show convergence. For this reason, the proof does not apply directly to the MMGRCPSP. It is necessary to prove that an extension of Sprecher's algorithm for generalized precedence is still guaranteed to converge.

The task at hand is to develop a solution algorithm for the subproblems of the decomposition method proposed in this dissertation. Since the subproblems are merely smaller instances of the complete MMGRCPSP being decomposed, the form of these subproblems is the same as that given in Table 3 and Figure 10. If the subproblems of the decomposition had standard precedence requirements, Sprecher's algorithm could be used as he originally developed it. Since this is not the

Table 3 MMGRCPSP Model Definitions

$A$	the set of all activities
$d$	the index of the terminal activity
$e_{im}$	the earliest start for activity $i$ in mode $m$
$l_{im}$	the latest start for activity $i$ in mode $m$
$M_i$	the set of all execution modes for activity $i$
$\tau_{im}$	the duration of activity $i$ in mode $m$
$S_i$	the set of generalized successors of activity $i$
$\Delta_{ijmn}$	the minimum lag between the start time of activity $i$ in mode $m$ and the start time of activity $j \in S_i$ in mode $n$
$K$	the set of all doubly constrained resources
$r_{imk}$	the number of units of resource $k$ required by activity $i$ when being executed in mode $m$
$R_k$	the per period availability of resource $k$
$N_k$	the total amount of resource $k$ available
$g$	the deadline for the project under consideration
$x_{imt}$	1 if activity $i$ starts in period $t$ using mode $m$

---


$$\text{Minimize } \sum_{m \in M_d} \sum_{t=e_{dm}}^{l_{dm}} (t + \tau_{dm}) x_{dmt} \quad (61)$$

$$\text{Subject to } \sum_{i \in A} \sum_{m \in M_i} \sum_{t=e_{im}}^{l_{im}} r_{imk} x_{imt} \leq N_k \quad \forall k \in K \quad (62)$$

$$\sum_{i \in A} \sum_{m \in M_i} \sum_{t=j-\tau_{im}+1}^j r_{imk} x_{imt} \leq R_k \quad \forall \left\{ \begin{array}{l} k \in K \\ j \in [1, g] \end{array} \right. \quad (63)$$

$$\sum_{n \in M_j} \sum_{t=e_{jn}}^{l_{jn}} (\Delta_{ijm'n} - t) x_{jnt} + \sum_{t=e_{im'}}^{l_{im'}} t x_{im't} \leq \sum_{m \in (M_i \setminus m')} \sum_{t=e_{im}}^{l_{im}} g x_{imt} \quad \forall \left\{ \begin{array}{l} i \in A \\ j \in S_i \\ m' \in M_i \end{array} \right. \quad (64)$$

$$\sum_{m \in M_i} \sum_{t=e_{im}}^{l_{im}} x_{imt} = 1 \quad \forall i \in A \quad (65)$$

$$x_{imt} \in \{0, 1\} \quad \forall \left\{ \begin{array}{l} i \in A \\ m \in M_i \\ t \in [e_{im}, l_{im}] \end{array} \right. \quad (66)$$


---

Figure 10 MMGRCPSP Model Formulation

case, Sprecher's algorithm must be extended to handle the generalized precedence relations found in this study's subproblems.

The extended algorithm presented here, like Sprecher's algorithm, implicitly enumerates every active permutation schedule for a given problem. To guarantee convergence, it is necessary to prove that for every feasible problem, there exists an active permutation schedule that is optimal for that problem. As long as the algorithm's bounding rules only prune dominated branches from the solution tree, the algorithm is guaranteed to converge.

There could be a problem establishing convergence for the formulation in Figure 10. The generalized precedence constraints expressed by Equation (64) allow for an activity  $i$  to have some other activity  $j$  as both a predecessor and a successor if the corresponding lags are both equal to zero. Such a precedence condition represents an activity pair  $(i,j)$  that is constrained to have simultaneous start times. While this is a valid constraint, its use in the problem formulation would make it possible for a feasible problem to have no active permutation schedule that is optimal. This is due to the fact that when constructing an active permutation schedule, the feasibility of an activity's start time is only checked with respect to its predecessors, not its successors.

To deal with the problem posed by the possibility of simultaneous start time constraints, the following restriction is placed on the generalized successor sets,  $S_i$ .

$$i \notin S_{j \in S_i} \quad \forall i \in A$$

This restriction prohibits an activity  $j$  from being both predecessor and successor to any other activity  $i$ . The disadvantage of imposing this restriction is that simultaneous start time constraints can no longer be modeled. It is possible to finesse this difficulty by modeling two or more activities with simultaneous start times as a single activity with aggregate resource requirements. In this way, the simultaneous start time condition can be modeled and the extended algorithm is still guaranteed to converge to optimality. With the simultaneous start time restriction in place, it is possible to prove that an extension of Sprecher's algorithm for generalized precedence constraints converges for any feasible problem instance.

The first step in proving convergence for an extended Sprecher algorithm is to prove the existence of an optimal active permutation schedule for every feasible problem instance. This is

Table 4 Existence Theorem

Theorem	If $P$ is a feasible MMGRCPSP with a regular performance measure for an objective function, then there exists an active permutation schedule that is optimal for $P$ .
Proof:	<p>Let <math>P</math> be a MMGRCPSP with objective function <math>\Phi</math>, a regular performance measure.</p> <p>If <math>P</math> is feasible, then <math>\exists</math> some schedule <math>S = (S, \mathcal{M})</math> that is optimal for <math>P</math>.</p> <p>Assume <math>S</math> is not an active permutation schedule and show that <math>\exists</math> an active permutation schedule <math>S' = (S', \mathcal{M})</math> such that <math>\Phi(S', \mathcal{M}) \leq \Phi(S, \mathcal{M})</math>.</p> <p>Set <math>S' = S \Rightarrow (ST'_i = ST_i \quad \forall i = 1, \dots, J)</math>.</p> <p>Let <math>PS_i, i = 1, \dots, J</math>, be the partial schedules that <math>S'</math> completes.</p> <p>Since <math>S'</math> is not an active permutation schedule, <math>\exists</math> at least one <math>PS_i</math> that is not an active partial schedule.</p> <p>Let <math>k</math> be the lowest index among the inactive partial schedules of <math>S'</math>.</p> <p>We know that <math>PS_k</math> is not an active partial schedule of <math>S'</math>.</p> <p><math>\Rightarrow \exists</math> at least one left shift (<math>ls</math>) for activity <math>k</math> in <math>PS_k</math>.</p> <p>Let <math>LS_k</math> be the set of all possible left shifts for activity <math>k</math> in <math>PS_k</math>.</p> <p>Set <math>ST'_k</math> to the minimum start time for activity <math>k</math> over all <math>ls \in LS_k</math>.</p> <p>We need to show that <math>S'</math> is still feasible after the left shift of activity <math>k</math>.</p> <p>To show feasibility, we must consider the following two cases.</p> <p>Case 1: Consider the feasibility of an activity <math>i</math> where <math>i \in PS_k</math>.</p> <p>A left shift was performed on activity <math>k</math> in partial schedule <math>PS_k</math>.</p> <p>By definition, a left shift operation results in a feasible schedule.</p> <p>Therefore, the resulting partial schedule, <math>PS'_k</math>, is feasible.</p> <p>Case 2: Consider the feasibility of an activity <math>i</math> where <math>i \notin PS_k</math>.</p> <p>Activity <math>i</math> was feasible in <math>S'</math> before the left shift to activity <math>k</math>.</p> <p>The only resulting schedule change is a new start time <math>ST''_k &lt; ST'_k</math>.</p> <p>Since <math>i \notin PS_k</math>, <math>i</math> is either a successor of <math>k</math> or they have no precedence relation.</p> <p>If <math>i \in S_k</math>, <math>ST'_k \leq ST'_i \Rightarrow ST''_k &lt; ST'_k \leq ST'_i</math>.</p> <p>Therefore, activity <math>i</math> is precedence feasible after the left shift.</p> <p>Let <math>O'</math> be the span of time that activities <math>i</math> and <math>k</math> overlap in <math>S'</math>.</p> <p>Let <math>O''</math> be the span of this overlap after the left shift of activity <math>k</math>.</p> <p>Since <math>ST''_k &lt; ST'_k \leq ST'_i</math> and activity durations are unchanged, <math>O'' \subset O'</math>.</p> <p>Activity <math>i</math> was resource feasible before the left shift and <math>O'' \subset O'</math>.</p> <p>Therefore, activity <math>i</math> is resource feasible after the left shift.</p> <p>Activity <math>i</math> is both precedence and resource feasible after the left shift.</p> <p>If there are still inactive partial schedules, repeat logic for the lowest index, <math>k</math>.</p> <p>When there are no more inactive <math>PS_i</math>, let <math>\mathcal{G} = (g_1, \dots, g_J)</math> be a permutation of all activities <math>1, \dots, J</math> such that <math>ST'_{g_i} \leq ST'_{g_{i+1}} \quad \forall i = 1, \dots, J-1</math>.</p> <p>The schedule <math>(S', \mathcal{M})</math> is a feasible active permutation schedule with permutation <math>\mathcal{G}</math>.</p> <p><math>\Phi</math> is a regular performance measure and <math>ST'_i \leq ST_i \quad \forall i \in A</math> so <math>\Phi(S', \mathcal{M}) \leq \Phi(S, \mathcal{M})</math>, but <math>S</math> is optimal so <math>\Phi(S', \mathcal{M}) \geq \Phi(S, \mathcal{M}) \Rightarrow \Phi(S', \mathcal{M}) = \Phi(S, \mathcal{M})</math>.</p> <p><math>\Rightarrow</math> Schedule <math>S' = (S', \mathcal{M})</math> is an optimal active permutation schedule for <math>P</math>.</p> <p>Therefore, <math>\exists</math> an optimal active permutation schedule for any feasible <math>P</math>.</p>

done by showing that for any feasible problem instance, it is possible to take an optimal schedule and transform it using left shift operators. The transformation results in an active permutation schedule that, due to the properties of regular measures of performance, has the same objective function value as the original schedule. Since the active permutation schedule resulting from the left shift transformation has the same objective function value as the original optimal schedule, it is an optimal active permutation schedule, and existence has been shown. The details of the existence proof are given in Table 4. After the extended algorithm is presented, a convergence proof is given.

*4.1.2 Preprocessing.* In the development of his algorithm, Sprecher focused primarily on problems with the objective of minimizing the overall project makespan. He used the latest activity start and finish times to show dominance and to fathom, or prune, branches in his search tree. The latest start and finish times for all activities are determined before the algorithm is begun and then adjusted each time a new solution is found. The initial values are found by setting the latest finish time for the terminal activity to an upper bound on project makespan and then making a CPM-type backward pass to obtain the latest start and finish times for all of the project activities.

In Sprecher's algorithm, every time a solution is found that is better than the current best solution, it represents a tighter upper bound on the optimal project makespan and the latest activity start and finish times are adjusted accordingly. Since these values are obtained without consideration for the availability of limited project resources, they are optimistic, but not necessarily feasible, estimates. However, they require very little computation time to generate, thus allowing quick evaluation of nodes in the search tree.

The extended Sprecher subproblem solution algorithm is applied to problems with objectives other than makespan, but the latest activity start and finish times still play an important role in establishing upper bounds for any regular performance measure. Although the straight-forward, CPM-based approach Sprecher uses to obtain these values does not apply once generalized precedence is considered, similar logic can be applied to develop an algorithm for generating latest activity start and finish times under generalized precedence. Table 5 provides definitions for some additional parameters that are required by the preprocessing algorithm that obtains these values.

Along with the definitions given in Table 5, the preprocessing algorithm requires several assumptions. First, the algorithm assumes that for each activity, the execution modes are stored



Table 5 Preprocessing Algorithm Definitions

$P_i$	the set of generalized predecessors of activity $i$
$\bar{T}$	the upper bound on the optimal project makespan
$LS_i$	the upper bound on the latest start time for activity $i$
$LF_i$	the upper bound on the latest finish time for activity $i$
$A^s$	the set of all scheduled (labeled) activities
$A^e$	the set of all eligible activities

in ascending order. This means that  $\tau_{i1}$  is the shortest possible duration for activity  $i$ . Second, it is assumed that activity 1 is the only activity that has no predecessors and that activity  $J$  is the only activity that has no successors. Dummy activities can be used to insure that this assumption is met. Finally, the initial upper bound on the optimal project makespan requires the assumption that  $\Delta_{ijmn} \leq \tau_{im}$  for all predecessor/successor pairs  $(i, j)$  and every possible combination of modes  $(m, n)$ . If this last assumption is not met, the algorithm is still applicable, but the initial upper bound on the optimal project makespan must be modified to insure that it covers the absolute worst case. This could be accomplished by increasing  $\bar{T}$  by  $\max_{m \in M_i, n \in M_j} \{\Delta_{ijmn} - \tau_{im}\}$  for every  $(i, j)$  predecessor/successor pair that violates the assumption that  $\Delta_{ijmn} \leq \tau_{im}$ .

Table 6 Preprocessing Algorithm

Step 1:	(Find an initial upper bound for the optimal project makespan) Set $\bar{T} = \sum_{i=1}^J \max_{m \in M_i} \{\tau_{im}\}$ .
Step 2:	(Initialize) Set $LF_J = \bar{T}$ , $LS_J = \bar{T} - \tau_{J1}$ , $A^s = \{J\}$ , and $A^e = P_J$ .
Step 3:	(Set the upper bounds for an eligible activity) Let $i$ be the highest activity index in the set $A^e$ . Set $LS_i = \max_{m \in M_i} \{\min_{j \in S_i} \{LS_j - \min_{n \in M_j} \{\Delta_{ijmn}\}\}\}$ . Set $LF_i = \max_{m \in M_i} \{\min_{j \in S_i} \{LS_j - \min_{n \in M_j} \{\Delta_{ijmn}\}\} + \tau_{im}\}$ .
Step 4:	(Update accounting sets and iterate until finished) Set $A^s = A^s \cup \{i\}$ and $A^e = \{A^e \setminus \{i\}\} \cup \{j : j \in P_i, S_j \subseteq A^s\}$ . If $i \neq 1$ , go to Step 3.

Table 6 shows the preprocessing algorithm that obtains latest start and finish times for every activity in a given project. Step 1 of the algorithm finds the worst case optimal project makespan by assuming that the nonrenewable resource levels are such that every activity must execute in its longest duration mode and the renewable resource levels prohibit any activities from executing concurrently. As mentioned previously, if  $\Delta_{ijmn} > \tau_{im}$  for any  $(i, j)$  predecessor/successor pairs,

the upper bound must be adjusted accordingly to be sure to account for the absolute worst case schedule.

The preprocessing algorithm makes a pass through a project's set of activities starting at the terminal activity and working backward, labeling an activity with latest start and finish times only after all of its successors have been labeled. Step 2 of the algorithm labels the terminal activity and initializes  $A^e$ , the set of activities that are eligible to be labeled, and  $A^s$ , the set of activities that have already been labeled. The latest finish time for the terminal activity is set to the upper bound on the optimal project makespan and the latest start time is obtained by subtracting the terminal activity's shortest duration time from its latest finish time. At initialization, set  $A^s$  includes only the terminal activity and the set  $A^e$  includes only those activities that are the immediate predecessors of the terminal activity.

The main iteration of the algorithm occurs in Steps 3 and 4 where some activity  $i$  is selected from the eligible set. Activity  $i$  is labeled with latest start and finish times and then moved from the eligible set to the labeled set. Any predecessors of activity  $i$  are then added to the eligible set if all of their successors have already been labeled. During each iteration, the latest start and finish times for activity  $i$  are found by considering all combinations of execution modes for activity  $i$  and its immediate successors. The algorithm terminates when all of the project's activities have been labeled.

*4.1.3 The Basic Algorithm.* Once the preprocessing algorithm (described in Table 6) has been run, the basic tree search algorithm implicitly enumerates every possible active permutation schedule in approximately the same manner as Sprecher's algorithm. The major differences are that data structures are added to keep track of the top  $k$  solutions encountered and that the bounding procedure accommodates both generalized precedence and regular measures of performance. The basic algorithm, described in Table 7, uses only a simple upper bound for pruning dominated branches from the search tree. Sprecher has developed several more complex bounding strategies that accelerate the execution of his algorithm. Some of these acceleration schemes are applicable to the extended algorithm and are addressed in later sections.

The extended algorithm shown in Table 7 implicitly enumerates every resource and precedence feasible active permutation schedule for a given MMGRCPSP. Since the goal of the algorithm is to

Table 7 Basic Extended Sprecher Algorithm

---

Step 1:	(Find an initial upper bound for the optimal solution) The bound that is used for pruning dominated branches is the value of the current $k^{th}$ best solution so obtain $k$ feasible solutions by either searching the tree or by use of some heuristic method.
Step 2:	(Initialize) Initialize the available renewable, nonrenewable, and doubly constrained resources. Let $i$ be the current level and set $i = 1$ . Let $g_i$ be the activity scheduled on level $i$ and set $g_1 = 1$ . Set $A^s = \emptyset$ , $A_1^e = \emptyset$ , and $m_{g_1} = 0$ . For completeness, set $g_0 = 0$ and $ST_{g_0} = 0$ .
Step 3:	(Identify the next branch to consider) If $m_{g_i} <  M_{g_i} $ , set $m_{g_i} = m_{g_i} + 1$ and go to Step 5. If $A_i^e \neq \emptyset$ , choose $g_i \in A_i^e$ , then set $A_i^e = A_i^e \setminus \{g_i\}$ , set $m_{g_i} = 1$ , and go to Step 5.
Step 4:	(Backtrack up one level) Set $i = i - 1$ and if $i = 0$ , STOP. Set $A^s = A^s \setminus \{g_i\}$ and adjust the available resources. Go to Step 3.
Step 5:	(Attempt to schedule activity $g_i$ in mode $m_{g_i}$ ) Let $ES_{g_i}$ be the earliest precedence feasible start time for activity $g_i$ in mode $m_{g_i}$ . Set $ES_{g_i} = \max(ST_{g_{i-1}}, \{ST_j + \Delta_{jg_i m_j m_{g_i}} : j \in P_{g_i}\})$ . Find $t$ , the earliest resource feasible start time for $g_i$ in mode $m_{g_i}$ such that $ES_{g_i} \leq t \leq LF_{g_i} - \tau_{i m_{g_i}}.$ If no $t \in [ES_{g_i}, LF_{g_i} - \tau_{i m_{g_i}}]$ is resource feasible or if, given $ST_{g_i} = t$ , the best completion to $PS_i$ is dominated by the current bound, go to Step 3. Set $ST_{g_i} = t$ , $A^s = A^s \cup \{g_i\}$ , and adjust the available resources. If $i = J$ , go to Step 7.
Step 6:	(Set up the next level) Set $i = i + 1$ and $A_i^e = \{j : j \notin A^s, P_j \subseteq A^s\}$ . Choose $g_i \in A_i^e$ , then set $A_i^e = A_i^e \setminus \{g_i\}$ and $m_{g_i} = 0$ . Go to Step 3.
Step 7:	(Store the new solution) Remove the $k^{th}$ best solution from the $k$ -best solution set and add the new solution. Set the upper bound for pruning to the value of the new $k^{th}$ best solution. Set $A^s = A^s \setminus \{g_i\}$ and adjust the available resources. Go to Step 3.

---

find the  $k$ -best solutions, the algorithm uses the objective function value of the  $k^{th}$  solution to prune dominated branches. In Step 1 of the algorithm,  $k$  feasible solutions can be externally supplied to allow the algorithm to begin pruning branches immediately. The better these  $k$  initial solutions, the tighter the initial bound, and the faster the algorithm can converge. If no initial solutions are supplied, the algorithm traverses the solution tree without pruning any branches until  $k$  feasible solutions have been observed and recorded. Once the  $k^{th}$  feasible solution has been found, pruning may begin.

In Step 2, the data structures that control the algorithm's execution are initialized. The variable  $i$  keeps track of what level of the solution tree the algorithm is currently investigating. As discussed previously, one activity is scheduled at each level so the greater the value of  $i$ , the more activities are included in the current partial schedule. The initial value of  $i$  is 1. The variable  $g_i$  identifies the activity that is scheduled at level  $i$  in the current partial schedule. It is assumed that activity 1 is the only activity without any predecessors so the initial value for  $g_1$  is 1. The variable  $m_i$  identifies the mode that activity  $g_i$  is scheduled in at level  $i$  of the current partial schedule. The set  $A^s$  is the set of activities included in the current partial schedule and is initially empty. The set  $A_i^e$  is the set of activities that are eligible for scheduling at level  $i$  in the current partial schedule and initially, the set  $A_1^e$  is empty.

At a given level of the solution tree,  $i$ , there is a fork for every mode of every activity in the set of eligible activities,  $A_i^e$ . The algorithm keeps track of which of these forks have been explored or pruned and which are candidates for further exploration. In Step 3, the algorithm selects the next unexplored/unpruned fork for branching, sets the variables  $g_i$  and  $m_{g_i}$  to the activity and mode number that correspond to the selected fork, and then moves on to Step 5. If all of the forks at the current level have been explored or pruned, the algorithm moves to Step 4 where activity  $g_i$  is removed from the set of scheduled activities, the associated resources are freed, and the algorithm moves up one level in the solution tree and goes back to Step 3. The algorithm cannot move up a level from level 1 so the algorithm terminates in Step 4 if the current level is 1.

Once a fork has been selected, the algorithm attempts to schedule activity  $g_i$  in mode  $m_{g_i}$  at level  $i$  in the current partial schedule. If a feasible start time cannot be found, the current fork is considered pruned and the algorithm returns to Step 3 to select a different fork to explore. If a

feasible start time can be found,  $ST_{g_i}$  is set equal to the earliest resource and precedence feasible start time for activity  $g_i$  in mode  $m_{g_i}$  at level  $i$  in the current partial schedule. If the most optimistic completion (MOC) of the current partial schedule (including activity  $g_i$ ) is bounded by the objective function value of the current  $k^{th}$  best solution, the the current fork is considered pruned and the algorithm returns to Step 3.

The process of finding the most optimistic completion of a partial schedule depends on the objective function being used. For a simple makespan objective function, the MOC can be expressed as  $\max_{j \in PS_i} \{ST_j + (LF_j - LS_j)\}$ . The other objective function that is used in this dissertation is makespan weighted by resource cost. The MOC for this objective function is found by adding the cost of the resources used by the activities in the current partial schedule and the cost of the resources for the most efficient mode of each activity not in the current partial schedule to the most optimistic makespan.

If activity  $g_i$  can be feasibly scheduled in mode  $m_{g_i}$  without being bounded, it is scheduled to start execution at  $ST_{g_i}$  at level  $i$  of the current partial schedule. Activity  $g_i$  is then added to the set  $A^s$  and the available resources are decremented according to the resources required by activity  $g_i$  in mode  $m_{g_i}$ . If the current partial schedule includes all of the project activities, the algorithm moves to Step 7 to store the new solution. If there are still activities that need to be scheduled, the algorithm goes to Step 6 to move a level deeper into the solution tree.

Moving a level deeper into the solution tree, Step 6 of the algorithm fills the data structures that contain all of the possible forks given the current partial solution. The variable  $i$  is incremented to reflect the new level and the set  $A_i^e$  is populated with every unscheduled activity for which all predecessors are already included in the current partial schedule. At level  $i$ , there is a fork for every execution mode of every activity included in the set  $A_i^e$ . After the forks have been set up, the algorithm moves on to Step 3 where a fork is selected to be explored.

When the current partial schedule gets to the point where it includes every project activity, a feasible solution has been found and the objective function value of that solution is at least better than the current  $k^{th}$  best solution. At this point, Step 7 of the algorithm inserts this new solution into the appropriate position in a sorted list of the current  $k$  best solutions. The solution that was previously ranked  $k^{th}$  is dropped from the list and the new  $k^{th}$  best objective function value becomes

the new bound for pruning dominated branches. After the new solution is recorded, activity  $g_i$  is removed from the current partial schedule, the resources it required are freed, and the algorithm returns to Step 3 where a new fork is selected for exploration.

By following the steps outlined in Table 7, the extended algorithm implicitly enumerates every active permutation schedule for a given MMGRCPSP problem instance. As the solution tree is traversed, the algorithm maintains a list of the top  $k$  solutions found and only prunes branches that are bounded by the objective function value of the current  $k^{th}$  best solution. It is already proven that there exists an optimal active permutation schedule for every feasible problem instance. To show convergence, it is now necessary to prove that the algorithm does not prune any nondominated solutions from the solution tree. The details of the convergence proof are given in Table 8.

Table 8 Optimality Theorem

---

Theorem	If $P$ is a feasible MMGRCPSP with a regular performance measure for an objective function, the best solution found by the extended Sprecher algorithm is an optimal solution for $P$ .
---------	---

---

Proof:

Let  $P$  be a feasible MMGRCPSP with objective function  $\Phi$ , a regular measure of performance.

$P$  must have an optimal active permutation schedule (Existence Theorem).

$\Rightarrow$  The best active permutation schedule is optimal for  $P$ .

We need to show that the extended Sprecher algorithm obtains the best active permutation schedule for  $P$ .

The algorithm tracks the best solution found, but not all solutions are examined.

Let  $S$  be the best active permutation schedule found by the extended algorithm.

Suppose  $\exists$  an active permutation schedule  $S'$  that is lower than  $S$ .

$\Rightarrow \Phi(S') < \Phi(S)$ .

If  $S'$  was not found by extended Sprecher,  $S'$  must be a completion of one of the partial schedules that was pruned from the solution tree.

But we know that the extended algorithm only prunes partial schedules if the best possible completion is dominated  $\Rightarrow \Phi(S') > \Phi(S)$ .

This is a contradiction.

Therefore, extended Sprecher must find the best active permutation schedule for  $P$ .

Since the best active permutation schedule is optimal, the extended algorithm has found the optimal solution.

---

**4.1.4 Acceleration Schemes.** Sprecher describes several additional bounding rules that can be implemented to reduce the computation time required to solve a given problem [79]. Some

of these rules do not apply to the MMGRCPSP. However, the following four bounding rules are directly applicable to the extended algorithm as outlined in the previous sections.

**Bounding Rule 1** *If at some level in the solution tree, say level  $i$ , some eligible activity  $g_i \in A_i^e$  cannot be feasibly scheduled in any mode, then there exists no feasible completion to the current partial schedule  $PS_i$ . For this reason, all forks at level  $i$  may be pruned and the algorithm can backtrack up one level in the solution tree.*

**Bounding Rule 2** *Consider the following two active partial schedules.*

$$PS_{i+2} = \begin{pmatrix} 1 & 2 & \cdots & i & i+1 & i+2 \\ g_1 & g_2 & \cdots & g_i & g_{i+1} & g_{i+2} \\ m_{g_1} & m_{g_2} & \cdots & m_{g_i} & m_{g_{i+1}} & m_{g_{i+2}} \\ ST_{g_1} & ST_{g_2} & \cdots & ST_{g_i} & ST_{g_{i+1}} & ST_{g_{i+2}} \end{pmatrix}$$

$$\overline{PS}_{i+2} = \begin{pmatrix} 1 & 2 & \cdots & i & i+1 & i+2 \\ \bar{g}_1 & \bar{g}_2 & \cdots & \bar{g}_i & \bar{g}_{i+1} & \bar{g}_{i+2} \\ \bar{m}_{\bar{g}_1} & \bar{m}_{\bar{g}_2} & \cdots & \bar{m}_{\bar{g}_i} & \bar{m}_{\bar{g}_{i+1}} & \bar{m}_{\bar{g}_{i+2}} \\ \overline{ST}_{\bar{g}_1} & \overline{ST}_{\bar{g}_2} & \cdots & \overline{ST}_{\bar{g}_i} & \overline{ST}_{\bar{g}_{i+1}} & \overline{ST}_{\bar{g}_{i+2}} \end{pmatrix}$$

Now, suppose that  $g_j = \bar{g}_j$ ,  $m_{g_j} = \bar{m}_{\bar{g}_j}$ , and  $ST_{g_j} = \overline{ST}_{\bar{g}_j}$  for all  $j = 1, \dots, i$ .

Suppose further that  $g_{i+1} = \bar{g}_{i+2}$ ,  $g_{i+2} = \bar{g}_{i+1}$ ,  $m_{g_{i+1}} = \bar{m}_{\bar{g}_{i+2}}$ ,  $m_{g_{i+2}} = \bar{m}_{\bar{g}_{i+1}}$ ,  $ST_{g_{i+1}} = \overline{ST}_{\bar{g}_{i+2}}$ , and  $ST_{g_{i+2}} = \overline{ST}_{\bar{g}_{i+1}}$ .

Given these conditions, the partial schedules  $PS_{i+2}$  and  $\overline{PS}_{i+2}$  are essentially the same. They include the same subset of activities, executing in the same modes, at the same times, and consuming the same resources. Therefore, they have the same set of feasible completions and if the algorithm encounters this situation, only one of the two forks needs to be explored.

**Bounding Rule 3** *Let  $i$  be the current level in the solution tree and let  $t$  be the earliest resource and precedence feasible start time for eligible activity  $g_i$ , in mode  $m_{g_i}$ , in the current partial schedule  $PS_i$ . If  $t < ST_{i-1}$ , then the fork associated with activity  $g_i$  and mode  $m_{g_i}$  at level  $i$  in the solution tree is dominated and may be pruned.*

**Bounding Rule 4** Let  $N_k(PS_i)$  be the nonrenewable quantity of resource  $k$  still available after scheduling the subset of activities included in partial schedule  $PS_i$ . Let  $\bar{A}^s$  be the set of activities that are not included in partial schedule  $PS_i$ . If  $N_k(PS_i) < \sum_{j \in \bar{A}^s} \min_{m \in M_j} \{r_{jmk}\}$  for any resource  $k \in K$ , then there is no resource feasible completion to the current partial schedule  $PS_i$  and the current fork may be pruned from the solution tree.

Table 9 shows how Step 5 of the basic extended algorithm is modified to implement all four of the new bounding rules along with the simple dominance rule used previously. Bounding Rules 3 and 4 are added with simple checks that require no additional algorithm data structures. For these two rules, the algorithm simply compares the earliest resource and precedence feasible start time for activity  $g_i$  with the start time of activity  $g_{i-1}$  in the current partial schedule.

Table 9 Step 5 for the Accelerated Algorithm

---

Step 5:	(Attempt to schedule activity $g_i$ in mode $m_{g_i}$ )
	Let $ES_{g_i}$ be the earliest precedence feasible start time for activity $g_i$ in mode $m_{g_i}$ .
	Set $ES_{g_i} = \max\{ST_j + \Delta_{jg_i m_j m_{g_i}} : j \in P_{g_i}\}$ .
	Find $t \in [ES_{g_i}, LF_{g_i} - \tau_{i m_{g_i}}]$ , the earliest resource feasible start for $g_i$ in mode $m_{g_i}$ .
	If no $t \in [ES_{g_i}, LF_{g_i} - \tau_{i m_{g_i}}]$ is resource feasible, set status of current fork to $-1$ .
	Then, if $g_i$ is not schedulable in any mode, go to Step 4, else go to Step 3.
	If $t < ST_{g_{i-1}}$ , set status of current fork to $-1$ and go to Step 3.
	If $t = ST_{g_{i-1}}$ and status of the fork for activity $g_i$ in mode $m_{g_i}$ at level $i - 1$ is 0, set status of current fork to $-1$ and go to Step 3.
	If there is no resource feasible completion of the current partial schedule, set status of current fork to $-1$ and go to Step 3.
	If, given $ST_{g_i} = t$ , the most optimistic completion for $PS_i$ is dominated, set status of current fork to $-1$ and go to Step 3.
	Set fork status to 1, $ST_{g_i} = t$ , $A^s = A^s \cup \{g_i\}$ , and adjust the available resources.
	If $i = J$ , go to Step 7.

---

If  $g_i$  can feasibly begin execution prior to the start time of activity  $g_{i-1}$ , then the conditions of Bounding Rule 3 have been met and the current fork may be pruned. If the earliest start for activity  $g_i$  is later than the start time for activity  $g_{i-1}$ , then the algorithm computes the minimum nonrenewable resource demand for the activities not yet included in the current partial schedule. If there is not enough available resources to meet this minimum demand, the conditions of Bounding Rule 4 have been met and the current fork may be pruned.

The mechanics of the basic algorithm require a set of possible forks be maintained at every level of the solution tree. Each fork is associated with an activity/mode combination that could



be scheduled at that level. Bounding Rules 1 and 2 require that one additional data element, fork status, be recorded for every fork. Each time the algorithm initializes a new level of the solution tree (Step 6) the status of all forks at this level is initialized to 0. When a fork is explored, its status is then changed to 1 if it was possible to schedule the activity/mode combination, or  $-1$  if the activity/mode combination could not be scheduled.

In the new Step 5 of the extended algorithm, whenever it is not possible to schedule an activity/mode combination, the algorithm considers the status of all of the forks associated with the same activity at that level. If the status of all of these forks is  $-1$ , it indicates that the activity cannot be scheduled in any mode at the current level. This meets the conditions of Bounding Rule 1 and allows the algorithm to backtrack up one level in the solution tree.

The conditions of Bounding Rule 2 are met in Step 5 if the earliest resource and precedence feasible start time for activity  $g_i$  in mode  $m_{g_i}$  is the same as the start time for activity  $g_{i-1}$  in the current partial schedule. When these conditions are met, the algorithm considers the status of the fork associated with activity  $g_i$ , in mode  $m_{g_i}$ , at level  $i - 1$ . According to Bounding Rule 2, that fork contains the same set of feasible solutions as the current fork. If the status of that fork is 0, it has yet to be explored and the current fork can be pruned.

**4.1.5 Testing.** In this section, the extended Sprecher algorithm is applied to a number of test problems to observe the computation time necessary to solve different problems instances and to determine how changes in problem parameters affect solution time. The algorithm is coded in C and the results are reported as solution time in seconds. The test runs are made on a 486DX66 processor with the C function *gettimeofday* used to mark the time the algorithm begins and ends execution. The solution time for each run is measured to the nearest one hundredth of a second.

The parameters used to describe specific problem instances are grouped into two distinct categories, size parameters and complexity parameters. The number of activities in a project directly affects the number of variables necessary to model the project. The density of the underlying precedence network in a project directly affects the number of constraints necessary to model a project. Since it is obvious that these parameters influence the size of the formulation necessary to model a project, they are good examples of size parameters.

It is relatively easy to postulate on the affect a change in a size parameter has on problem solution time. Later sections focus on how large a problem instance may be before solution time becomes prohibitive. The testing in this section, however, is more concerned with applying the algorithm to problem instances of a standard size and determining which instances take the most time to solve. With this in mind, the sets of test problems are constructed to allow for the study of the three significant complexity parameters identified by Sprecher: resource factor (*RF*), periodic resource strength (*PRS*), and total resource strength (*TRS*) [80, 81]. These three parameters together quantify the demand for and availability of project resources.

The problem generator ProGen allows these parameters to be set to any value in the range  $[0, 1]$  and generates a number of sample problems with resource complexity within a given tolerance of the values specified [51, 50]. For simplicity of illustration, assume that every activity in a given project has the same total number of execution modes. Let  $J$  be the number of activities in a project, let  $K$  be the number of resources, and let  $M$  be the number of modes for each activity. The resource factor parameter is defined as follows.

$$RF = \frac{1}{J} \frac{1}{K} \frac{1}{M} \sum_{i=1}^J \sum_{k=1}^K \sum_{m=1}^M \begin{cases} 1 & \text{if } r_{imk} > 0 \\ 0 & \text{otherwise} \end{cases}$$

The equation for the complexity parameter *RF* clearly shows that this parameter is a measure of the number of different resources required by the average activity/mode combination. A resource factor of 1 indicates that every mode for every activity requires a nonzero quantity of every project resource while a resource factor of 0 indicates that every mode for every activity requires absolutely no project resources. ProGen accepts any value in the range  $[0, 1]$  for this parameter and randomly sets, then iteratively adjusts, the resource demand values,  $r_{imk}$ , until the observed resource factor is within some acceptable tolerance of the value specified.

The other two complexity parameters of interest, *PRS* and *TRS*, are both measures of resource availability for a given set of resource demand values,  $r_{imk}$ . The difference between the two is that *PRS* quantifies the periodic resource availability while *TRS* quantifies the total resource availability. Based on the resource demand values it has already generated, ProGen derives values for the minimum and maximum resource availability, both periodic and total, for every project

resource. Let  $R_k^{min}$ ,  $R_k^{max}$ , and  $R_k$ , be the minimum, maximum, and actual periodic resource availability for project resource  $k$  and let  $N_k^{min}$ ,  $N_k^{max}$ , and  $N_k$ , be the minimum, maximum, and actual total resource availability for project resource  $k$ . Given these definitions, the equations for the complexity parameters  $PRS$  and  $TRS$  are given as follows [51, 50].

$$PRS = \frac{R_k - R_k^{min}}{R_k^{max} - R_k^{min}}$$

$$TRS = \frac{N_k - N_k^{min}}{N_k^{max} - N_k^{min}}$$

The problem instances generated by ProGen may be either resource rich or resource poor depending on the values specified for these complexity measures (periodic resource availability and total resource availability). As the actual availability gets close to the maximum, resource strength gets close to 1, and as availability approaches the minimum, resource strength goes to 0. ProGen accepts values in the range  $[0, 1]$  for each of these parameters, setting the resource availability in the problem instances it generates accordingly. Renewable resource availability is only affected by the parameter  $PRS$ , and nonrenewable resource availability only uses  $TRS$ , but doubly constrained resource availability is affected by both.

To illustrate a MMGRCPSP instance and its  $k$ -best solutions, the first set of results report on the application of the extended Sprecher algorithm to one small sample problem. The data given in Tables 10 and 11 completely describe the sample problem that is used.

The sample problem was generated to have properties similar to a problem instance from Sprecher's test set, although activity durations were reduced in order to make the IP formulation small enough for CPLEX to solve in a reasonable amount of time. Each activity in the sample problem has three execution modes, with a duration of either 2, 3, or 4. Every activity execution mode requires either 2, 3, or 4 units of one of the projects three doubly constrained resources. Activity 2 is the only activity with no predecessors and activity 11 is the only activity with no successors. The original problem, as generated by ProGen, had a super-source activity 1 and a super-sink activity 12. These activities are unnecessary in single project problems and were removed from the sample problem.

Table 10 Precedence Information for the Sample Problem

Pred	Succ	$\Delta_{ij11}$	$\Delta_{ij12}$	$\Delta_{ij13}$	$\Delta_{ij21}$	$\Delta_{ij22}$	$\Delta_{ij23}$	$\Delta_{ij31}$	$\Delta_{ij32}$	$\Delta_{ij33}$
2	3	2	1	1	2	3	1	3	1	2
	5	1	1	1	1	2	3	3	4	3
	6	2	2	2	1	2	1	2	4	3
3	4	1	2	2	3	3	3	3	4	3
	8	1	1	1	3	2	2	4	1	1
	9	2	2	1	3	3	3	1	2	4
4	7	1	2	1	3	1	1	2	4	3
5	8	1	2	1	2	1	2	2	4	2
	9	1	1	1	1	2	3	1	2	4
6	7	1	1	1	1	1	3	1	3	2
	8	2	2	2	1	3	3	4	2	3
	9	2	1	1	1	3	3	1	1	2
7	11	2	1	2	3	1	3	2	2	4
8	10	2	2	2	2	2	1	2	1	4
9	10	2	2	1	2	3	1	2	1	1
10	11	2	2	2	2	3	3	2	2	3

Table 11 Execution Mode Information for the Sample Problem

Job	2			3			4			5			6		
Mode	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
Dur	2	3	4	2	3	4	2	3	4	2	3	4	2	3	4
D1	0	2	0	3	0	0	0	0	3	0	2	0	0	2	0
D2	0	0	4	0	2	0	4	0	0	0	0	3	0	0	2
D3	2	0	0	0	0	4	0	4	0	3	0	0	4	0	0
Job	7			8			9			10			11		
Mode	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
Dur	2	3	4	2	3	4	2	3	4	2	3	4	2	3	4
D1	4	0	0	0	0	3	0	2	0	0	3	0	4	0	0
D2	0	0	3	0	3	0	3	0	0	3	0	0	0	0	3
D3	0	2	0	4	0	0	0	0	3	0	0	2	0	4	0

The first doubly constrained resource, D1, has 4 units available per period and 14 total units available. For D2, these numbers are 5 and 15, and for D3, 6 and 16. Since ProGen generates problem instances with standard precedence relations, the original problem was modified to include the lag values shown in Table 10. These values are randomly generated integers restricted to be in the range from 1 to the duration of the predecessor activity. The objective function for the problem is simple makespan.

Table 12 *K*-Best Solution Set for the Sample Problem

Rank	Obj	$s/m_2$	$s/m_3$	$s/m_4$	$s/m_5$	$s/m_6$	$s/m_7$	$s/m_8$	$s/m_9$	$s/m_{10}$	$s/m_{11}$
1	10	0/1	1/2	4/1	1/1	2/2	6/3	4/1	5/2	6/3	8/1
2	10	0/1	1/2	4/1	1/2	2/1	6/3	4/1	4/2	6/3	8/1
3	10	0/1	1/2	4/2	1/2	2/1	5/2	4/3	4/1	6/1	8/1
4	10	0/1	1/2	4/1	1/2	2/2	6/3	4/1	5/2	6/3	8/1
5	10	0/2	1/3	5/2	2/2	1/3	6/3	3/2	4/2	5/3	8/1
6	11	0/1	2/1	3/1	4/1	2/1	4/1	6/2	6/2	7/3	9/1
7	11	0/1	2/1	3/1	4/1	2/1	4/1	6/2	5/3	7/3	9/1
8	11	0/1	2/1	3/1	4/1	2/1	5/2	5/3	5/1	7/1	9/1
9	11	0/1	2/1	3/1	4/1	2/1	5/2	6/2	5/2	7/3	9/1
10	11	0/1	2/1	3/1	4/1	2/1	6/2	5/3	5/3	7/1	9/1
†	10	0/2	1/3	5/2	2/2	1/3	6/3	3/2	4/2	6/3	8/1

The extended Sprecher algorithm took 1.19 seconds on a 486DX66 to find the ten best active permutation schedules for the sample problem. The schedules and their resulting makespans are given in Table 12. In Table 12, the headings  $s/m_i$  are shorthand for  $ST_i/m_i$ , so each column entry gives the start time and execution mode for activity  $i$  in one of the  $k$ -best solutions. The problem was also formulated as an integer program and passed to CPLEX along with information about the problem's special ordered sets and the best solution found by the extended Sprecher algorithm. The problem formulation consists of 1200 binary variables and 312 constraints. The mixed integer program solver in CPLEX required 32462.13 seconds on a Sun SPARCstation-10 to find the optimal solution for this problem. The schedule corresponding to this solution is shown on the bottom row of Table 12 with a † symbol in its rank column.

The solution identified by CPLEX is very similar to solution 5 found by the extended Sprecher algorithm. The only difference between the two solutions is a one time unit difference in the start time of activity 10. This difference illustrates the fact that the extended Sprecher algorithm searches out the top  $k$  active permutation schedules, but for every active schedule, there can be any number

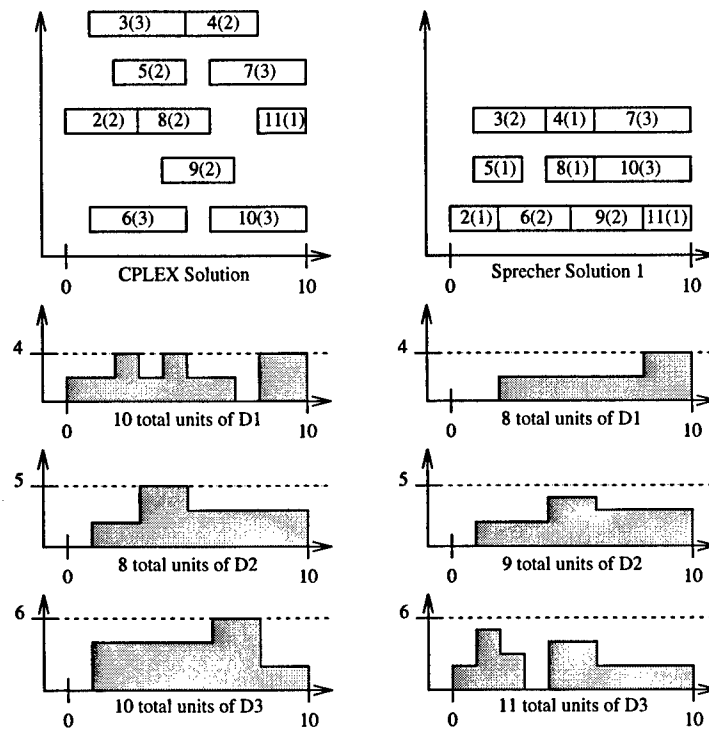


Figure 11 A Comparison of Two Alternate Optimal Solutions

of nonactive schedules with the same mode assignment and objective function value, but different activity start times. The CPLEX solution and solution 1 of the ten solutions found by the extended Sprecher algorithm are illustrated graphically in Figure 11.

The graphical illustration of these two alternate optimal schedules relates considerable information at a glance. The Gantt charts show the mode and start time for every project activity in each schedule. They also show that, for this example, although the project makespans are identical, there is more concurrent execution of activities in the solution from the extended Sprecher algorithm. The other charts in the figure provide an in depth analysis of the resource utilization for each solution. This sort of analysis could prove useful in selecting between alternate optimal solutions.

The next set of test results reports the performance of the algorithm in terms of average solution times on a set of sample problems generated to be roughly comparable to the main set of test problems used by Sprecher and Drexl [80, 81]. Tables 13 and 14 list the ProGen parameters that Sprecher and Drexl use to generate the problem instances for their test set. Recall, however, these problems used standard end to start precedence relations and only singly constrained resources.

Table 13 Constant Parameters Used By Sprecher and Drexl

	$J$	$ M_i $	$\tau_{im}$	$ R $	$U_R$	$Q_R$	$ N $	$U_N$	$Q_N$	$ S_1 $	$ S_j $	$ P_j $	$ P_j $
min	10	3	1	2	1	1	2	1	1	3	1	3	1
max	10	3	10	2	10	2	2	10	2	3	3	3	3

Table 14 Variable Parameters Used By Sprecher and Drexl

Parameter	Levels			
$RF_R$	0.5	1.0		
$RS_R$	0.2	0.5	0.7	1.0
$RF_N$	0.5	1.0		
$RS_N$	0.2	0.5	0.7	1.0

Table 13 uses some parameters that are now defined. The parameter  $R$  represents the set of renewable project resources while the parameter  $N$  is the set of nonrenewable resources. The minimum and maximum values for the parameters  $U_R$  and  $U_N$  provide bounds on the amount of a renewable or nonrenewable resource that may be requested for a single mode of any project activity. The parameters  $Q_R$  and  $Q_N$  restrict the number of different renewable or nonrenewable resources that may be required by a given activity mode. Table 14 shows the various levels of renewable and nonrenewable resource factor and resource strength that Sprecher and Drexl use to generate their test problems. The resource strength for the renewable resources,  $RS_R$ , is periodic resource strength, while the resource strength for the nonrenewable resources,  $RS_N$ , is total resource strength.

Sprecher and Drexl use only singly constrained resources, either renewable or nonrenewable, in their test problems. The combat modeling application presented in this dissertation is modeled using doubly constrained resources. Because of the requirements of the combat planning application, it is important that the problems used to test the extended Sprecher algorithm use doubly constrained resources.

All of the problem instances in the test set used by Sprecher and Drexl have four project resources: two renewable resources and two nonrenewable resources. For a given problem instance, it would take the same number of constraints to model one doubly constrained resource as it would to model one renewable and one nonrenewable resource. If the four singly constrained resources in a problem instance from Sprecher and Drexl's test set were replaced with two doubly constrained resources, to model the resulting problem instance would require the exact same number of variables and constraints, although the resource usage would vary and this could effect results.

In an effort to construct a set of test problems comparable to those used by Sprecher and Drexler, a decision had to be made regarding how many doubly constrained resources should be used. Using two doubly constrained resources would provide test problems with the same number of variables and constraints, but an argument could be made for using four, to ensure the same total number of resources. The set of test problems generated here to test the extended Sprecher algorithm used a compromise value of three doubly constrained resources for every problem instance. This is reasonable since the goal here is not to compare the performance of the original and extended algorithms, but to show that the effects of varying resource parameters are similar for each algorithm.

Table 15 Constant Parameters For the First Test Set

	$J$	$ M_i $	$\tau_{im}$	$ K $	$U_K$	$Q_K$	$ S_1 $	$ S_j $	$ P_J $	$ P_j $
min	10	3	1	3	1	1	3	1	3	1
max	10	3	10	3	10	3	3	3	3	3

Table 16 Variable Parameters For the First Test Set

Parameter	Levels		
$RF$	0.5	1.0	
$PRS$	0.5	0.7	1.0
$TRS$	0.5	0.7	1.0

Tables 15 and 16 show the ProGen parameters that are used to generate the first set of test problems for the extended Sprecher algorithm. Using doubly constrained resources in the test problems affects the variable parameters as well as the constant parameters. Sprecher and Drexler vary the resource factor and resource strength of both renewable and nonrenewable resources. For problems with doubly constrained resources, both the periodic and total resource strength are varied, but only one resource factor is necessary.

Sprecher and Drexler note that when a resource strength parameter is set to 0.2, some problem instances have no feasible solution. This behavior is a result of the manner in which ProGen implements resource strength [51]. In the set of test problems for the extended algorithm, almost all of the problem instances with a resource strength set to 0.2 have no feasible solution. It would appear that the doubly constrained nature of the resources makes the problems more sensitive to limited resources. For this reason, the 0.2 level of variable parameters  $PRS$  and  $TRS$  were eliminated from consideration in the generation of test problems.



The set of test problems was created using ProGen to generate 10 problem instances for every combination of levels of the variable parameters given in Table 16. There are 18 ( $2 * 3 * 3$ ) combinations of variable parameter levels, making the total number of problem instances in the test set 180. Generalized precedence lag values were randomly generated for each problem instance in the same manner as described previously for the sample problem. The extended algorithm was applied to every problem instance in the set and the resulting solution times are reported in Table 17.

Table 17 Solution Times For the First Test Set

Parameter	Level	Instances	Avg	Var	Max
RF	0.5	90	0.80	0.39	3.13
	1.0	90	3.37	12.90	25.56
TRS	0.5	60	2.98	9.03	11.62
	0.7	60	1.94	12.17	25.56
	1.0	60	1.32	2.45	9.17
PRS	0.5	60	2.89	8.52	11.60
	0.7	60	2.22	13.21	25.56
	1.0	60	1.14	1.76	6.55

The results reported in Table 17 show how average solution time is affected by changing each of the complexity parameters previously identified. Resource factor,  $RF$ , is a measure of resource demand. Higher levels of  $RF$  indicate that individual activities require more project resources. The average solution time results show that problems are harder to solve when resource demand is high. Resource strength,  $RS$ , is a measure of resource availability, or supply. Higher levels of  $RS$  indicate that there are greater quantities of resources available to the project. The results show that problems are harder to solve when supply is low. The general observation from the average solution time results is that high demand and low supply are both factors that lead to MMGRCPSP instances that are more difficult to solve.

Another interesting observation about problem instance complexity is that, while the lowest average solution times are found when resource strengths are at 1.0, there is more variance in solution times when availability is moderate or low ( $RS = 0.7$  or  $RS = 0.5$ ). To illustrate, observe that the problem instance with the largest solution time occurred not at the lowest levels of resource strength, but at the moderate levels ( $TRS = PRS = 0.7$ ). It would appear that problem instances with high resource availability are consistently the the easiest to solve. Instances with low availability

are generally the hardest, but due to increased variance, solution time is harder to predict when resource availability is low or moderate.

The activity modes in the problem instances that make up the first set of test problems do not accurately reflect the modes that are used in the type of combat planning problems presented in this dissertation. To keep the test set comparable to the one used by Sprecher and Drexler, activity modes were allowed to request more than one type of resource. In the combat planning problem, each resource represents a type of aircraft located at a base and the activities are targets that need to be attacked. Each mode of an activity represents a different aircraft type/location that could be used to attack the target (4 F-16 from base A or 2 F-111 from base B for example). For this reason, every mode in combat planning problem instances requires some quantity of exactly one project resource.

A second set of test problems was generated in the same manner as the first, but with the additional requirement that every activity mode use some nonzero quantity of exactly one project resource. The set was generated with ProGen using the same levels of the complexity parameters and the same number of instances for each combination of parameter levels. The additional requirement, however, forces the parameter  $RF = 0.33$  for every problem instance regardless of the parameter level set in ProGen. For this reason, the set contains the same total number of observations as before, but only two of the complexity parameters actually change level in the second set of test problems.

Table 18 Second Test Set Solution Times

		PRS			Average
		0.50	0.70	1.00	
TRS	0.50	0.42	0.37	0.33	0.38
	0.70	0.48	0.26	0.26	0.33
	1.00	0.40	0.35	0.19	0.31
Average		0.43	0.33	0.26	0.34

The extended algorithm was used to solve every problem instance in this second set and the resulting average solution times are reported in Table 18. The reason for running the algorithm on the second set of test problems was to gauge how difficult combat planning problem instances are compared to the types of problem instances observed in the first test set. The conclusion is that the restriction on the number of resources that an activity mode may request leads to problem

instances with smaller solution times. This observation follows the trend observed in the first set of test problems, where results showed that lower resource demand meant smaller solution times.

*4.1.6 Complexity.* In the initial testing of the extended Sprecher subproblem solution algorithm, the algorithm was applied to a small sample problem. This same problem was then solved with the only possible alternative solution method, a standard integer program solver. To improve the solution time of the IP solver, the special ordered sets were identified before execution, and the extended Sprecher optimal solution was used for an initial bound. The extended Sprecher algorithm was able to obtain an optimal schedule in about a second, while the standard IP solver in CPLEX took over nine hours to converge, even starting from a known optimal solution.

In this section, complexity analysis is presented to offer a theoretical explanation for the overwhelming difference between the solution times of the extended Sprecher algorithm and those of the standard IP solver. Both methods perform an implicit enumeration of solutions by traversing a tree structure, but the number of solutions on the tree and the quantity of work performed at each node differ. The complexity analysis presented here investigates each of these issues separately and offers evidence to show that the extended Sprecher algorithm is superior to the standard solver in both areas. The parameters in Table 19 are used throughout the complexity discussion.

Table 19 Complexity Parameters

$A$	the number of activities in the project
$M$	the number of modes for each activity
$T$	the number of periods in the planning horizon
$m$	the number of problem constraints
$n$	the number of problem variables ( $n = AMT$ )

The first issue to consider is the total number of solutions that each algorithm may be forced to enumerate in order to achieve optimality. Normally, for a problem with  $n$  binary variables, the number of possible combinations is given by  $2^n$ , but the presence of special ordered sets significantly reduces this number. In the MMGRCPSP formulation, there is one special ordered set for each activity and only one variable in each set can take on a value of 1 in a feasible solution. Taking advantage of this property, the number of possible solutions the IP solver may have to enumerate is reduced to  $(TM)^A$  which is significantly smaller than  $2^{AMT}$ .

The extended Sprecher algorithm further reduces the number of possible solutions by only enumerating those solutions that represent active schedules. The extended Sprecher algorithm implicitly enumerates every active permutation schedule. Since there could be a different active permutation schedule for every permutation of activities and combination of modes, the number of possible solutions is given by  $A!M^A$ . The quantity  $M^A$  can be factored out of the total number of possible solutions for both the standard solver and the extended Sprecher algorithm. The difference in the number of solutions these algorithms may have to enumerate is explained by the difference between the remaining terms;  $A!$  for extended Sprecher and  $T^A$  for the standard IP solver.

Since the parameter  $T$  does not appear in the term for the number of possible solutions for extended Sprecher, it is removed by focusing on the case that leads to the smallest number of possible solutions for the standard IP solver. The planning horizon,  $T$ , is a function of the durations of the project activities. The most favorable case, in terms of the smallest number of possible solutions, occurs when every mode for every activity has a duration of one period ( $T = A$ ). In this case, the number of possible solutions for the standard IP solver reduces to  $(AM)^A$ . Now, the quantity  $M^A$  may still be factored from each term and the difference is seen in the remaining terms;  $A!$  for extended Sprecher and  $A^A$  for the standard IP solver.

Figure 12 provides a graphical comparison of the complexity of the extended Sprecher algorithm and the standard IP solver. There are three problem parameters that affect the number of possible solutions for a given problem instance; the number of activities ( $A$ ), the number of modes ( $M$ ), and the number of periods in the planning horizon ( $T$ ). In Figure 12, the planning horizon is held constant at  $T = A$  while the number of activities and the number of modes per activity are varied from 1 to 10. It is obvious that, even using the planning horizon that is most favorable for the complexity of the standard IP solver, the extended Sprecher algorithm has a consistently smaller set of solutions to implicitly enumerate. The size difference between the solution spaces becomes more pronounced when more realistic planning horizons are considered.

It has been shown that, for a given problem instance, the extended Sprecher algorithm consistently has a smaller solution tree to traverse. This is important, but it is also important to consider the quantity of work it takes each algorithm to traverse that tree. To investigate this aspect of complexity, consider the tasks performed by each algorithm at every node within the solution tree.

At every node, the standard IP solver must solve an LP-relaxation of the original IP problem. This is typically accomplished with some variant of the traditional Simplex algorithm.

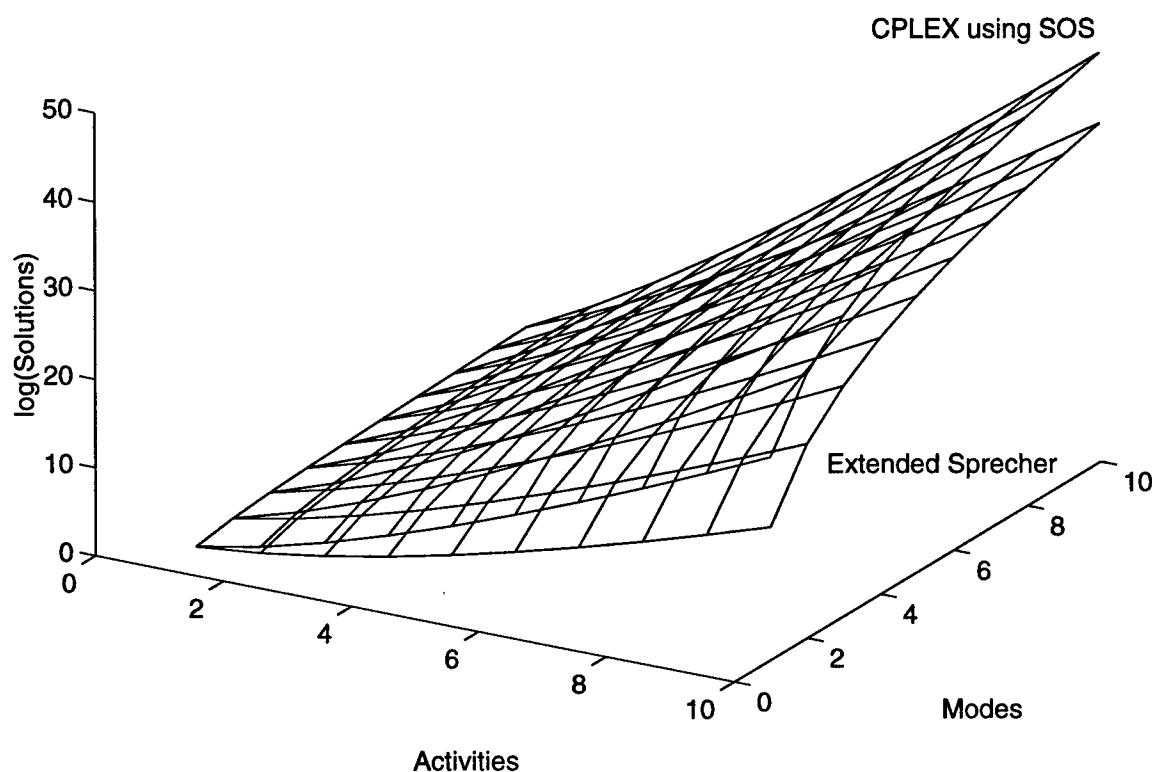


Figure 12 Theoretical Complexity Comparison

The Simplex algorithm solves linear programming problems by moving from one extreme point of the polyhedral feasible region of an LP problem to the next until an optimal point is found. Although the Simplex algorithm typically executes in polynomial time, it has been shown to require exponential effort for certain classes of difficult problems [6]. This is due to the fact that the algorithm could be forced to visit a total of  $\binom{n}{m}$  extreme points before it finds the optimal solution. This indicates that the order of the algorithm is  $O(\binom{n}{m})$ . For this reason, the worst case effort required by the standard IP solver at each node in the solution tree is  $\binom{n}{m}$ .

The extended Sprecher algorithm, even at its worst, requires far less computational effort at each node within the solution tree. At each node, extended Sprecher must attempt to add one

activity to the current partial schedule. The algorithm selects an activity and attempts to schedule it at the earliest possible time. The possible start times for any activity are limited by the number of periods in the planning horizon so at most  $T$  possible start times are considered. To evaluate the feasibility of a start time for an activity, the algorithm must check for constraint violations. Since there are  $m$  constraints, the maximum computational effort required at any given node in the solution tree is  $O(Tm)$ .

The order of the Simplex algorithm is  $O(\binom{n}{m})$  and the order of the work required by the extended Sprecher algorithm at each node visited is  $O(Tm)$ . It may be confusing to compare these values since one of them is a linear function and the other involves permutations. It is possible to obtain a lower bound for  $\binom{n}{m}$  as follows.

$$\binom{n}{m} = \frac{n!}{m!(n-m)!} = \left(\frac{n}{m}\right) \left(\frac{n-1}{m-1}\right) \left(\frac{n-2}{m-2}\right) \cdots \left[\frac{n-(m-1)}{m-(m-1)}\right] > \left(\frac{n}{m}\right)^m$$

Using this lower bound, it is apparent that for most values of  $n$  and  $m$  the worst case node effort for the standard IP solver dominates the worst case node effort for the extended Sprecher algorithm.

$$\left(\frac{n}{m}\right)^m > Tm$$

The IP solver applies the simplex algorithm at every node, which typically means polynomial effort but can require exponential effort, while the extended Sprecher algorithm only requires linear computational effort at every node. For any realistic problem, the worst case computational effort required by the extended Sprecher algorithm at each node is much less than that required by the standard IP solver. The smaller solution space and faster node evaluation combine to make the extended Sprecher algorithm far superior to a standard IP solver for solving the subproblems of the decomposition proposed in this dissertation. This observation is supported by the test results previously reported.

#### 4.2 Summary

This chapter presented the subproblem solution algorithm for the decomposition approach developed in this dissertation. The algorithm was developed by extending a solution approach for

a similar problem found in the literature. Convergence of the resulting algorithm was proven and empirical results were obtained for a range of representative problem instances. Chapter V uses the subproblem solution algorithm from this chapter to develop a Sweeney-Murphy integer program decomposition approach for large problem instances with block-angular structure.

## V. A Decomposition Algorithm

Hierarchical program and project management problems, or problems that may be modeled with a project management framework, exhibit classic block-angular problem structure. The block-angular structure present in the formulation of such problems indicates the presence of semi-independent subproblems within a given problem instance. These subproblems might be actual subsets of a single project, projects in a larger program, different phases of an acquisition program, or waves of combat missions in an ATO. In any case, the block-angular structure may be exploited by a decomposition algorithm to reduce the computational effort required to solve large problem instances.

In Chapter IV, the latest available techniques are extended to develop a specialized algorithm for the MMGRCPSP. While the testing showed the specialized algorithm developed in Chapter IV can solve MMGRCPSP instances faster than any approach currently reported, the order of the algorithm is still exponential and, like all other approaches, solving large problem instances quickly becomes computationally prohibitive. In this chapter, Sweeney-Murphy decomposition is implemented for the MMGRCPSP [84]. The goal is to use decomposition to further increase the size of MMGRCPSP instances that can be realistically solved.

### 5.1 Sweeney-Murphy Decomposition

Sweeney-Murphy decomposition extends the Dantzig-Wolfe method of linear program decomposition to integer programming. The Sweeney-Murphy algorithm is designed for integer program problem instances with the block-angular structure of problem **P**, illustrated in Table 20.

Table 20    A Block-Angular Integer Program

---

	Minimize	$c_1x_1$	+	$c_2x_2$	+	...	+	$c_px_p$	
	Subject to	$A_1x_1$	+	$A_2x_2$	+	...	+	$A_px_p$	$\geq b_0$
		$B_1x_1$							$\geq b_1$
(P)				$B_2x_2$					$\geq b_2$
						...			
								$B_px_p$	$\geq b_p$
								$x_1, x_2, \dots, x_p$	$\in Z^+$

---



Note that in Table 20, the notation  $Z^+$  is used to represent the set of all nonnegative integers. The semi-independent subproblems in problem **P** are bound together by the coupling constraints  $\sum_{i=1}^p A_i x_i \geq b_0$ . Sweeney and Murphy apply standard Lagrangian relaxation to the coupling constraints to allow problem **P** to be decomposed into  $p$  independent subproblems,  $\text{SP}_i(\mu)$ . The formulation of the Sweeney-Murphy subproblems is given in Table 21.

Table 21    The Sweeney-Murphy Subproblems

$$\begin{array}{ll} \text{SP}_i(\mu) & \begin{array}{l} \text{Minimize } (c_i - \mu A_i)x_i \\ \text{Subject to } \begin{array}{l} B_i x_i \geq b_i \\ x_i \in Z^+ \end{array} \end{array} \end{array}$$


---

Dantzig-Wolfe decomposition for linear programs operates by iteratively solving reformulated subproblems and a master problem to find the optimal convex combination of subproblem solutions. In integer program decomposition, convex combinations of subproblem solutions could result in non-integer values being assigned to variables that are supposed to be integer. With this in mind, the Sweeney-Murphy master problem formulation is constrained to select exactly one solution from the  $k$ -best solution set of each subproblem. The optimal solution to the master problem is the best combination of subproblem solutions that is feasible with respect to the previously relaxed coupling constraints.

For every subproblem  $i$ , it is necessary to search the subproblem solution space for the  $k_i$  solutions with the best objective function values. The resulting sets of  $k_i$ -best subproblem solutions are required to formulate the Sweeney-Murphy master problem. Let  $x_i^j$  represent the  $j^{\text{th}}$  best solution to subproblem  $i$  and let  $\lambda_{ij}$  be a binary variable in the master problem associated with the selection of the  $j^{\text{th}}$  best solution for subproblem  $i$ . If  $\lambda_{ij} = 1$ , then the master problem selects the  $j^{\text{th}}$  best solution for subproblem  $i$ . Since the master problem can only select one solution for each subproblem, only one  $\lambda_{ij}$  can take on a value of one for each subproblem  $i$ . This is controlled by a special ordered set of binary variables for each subproblem. These special ordered sets can be exploited to reduce solution time for the master problem.

The parameters for the objective function and the coupling constraints in the master problem are obtained by combining the parameters from the original problem **P** with the vectors of the  $k_i$  best

subproblem solutions. The objective function parameters are given by  $f_{ij} = c_i x_i^j$  and the coupling constraint parameters are given by  $q_{ij} = A_i x_i^j$ . The complete formulation of the Sweeney-Murphy master problem, **MP**, is shown in Table 22.

Table 22 The Sweeney-Murphy Master Problem

$$\begin{array}{ll}
 \text{Minimize} & \sum_{j=1}^{k_1} f_{1j} \lambda_{1j} + \sum_{j=1}^{k_2} f_{2j} \lambda_{2j} + \cdots + \sum_{j=1}^{k_p} f_{pj} \lambda_{pj} \\
 \text{Subject to} & \sum_{j=1}^{k_1} q_{1j} \lambda_{1j} + \sum_{j=1}^{k_2} q_{2j} \lambda_{2j} + \cdots + \sum_{j=1}^{k_p} q_{pj} \lambda_{pj} \geq b_0 \\
 & \sum_{j=1}^{k_1} \lambda_{1j} = 1 \\
 & \sum_{j=1}^{k_2} \lambda_{2j} = 1 \\
 & \vdots \\
 & \sum_{j=1}^{k_p} \lambda_{pj} = 1 \\
 & \lambda_{ij} \in \{0, 1\} \quad \forall i = 1, 2, \dots, p \quad j = 1, 2, \dots, k_i
 \end{array}$$

(MP)

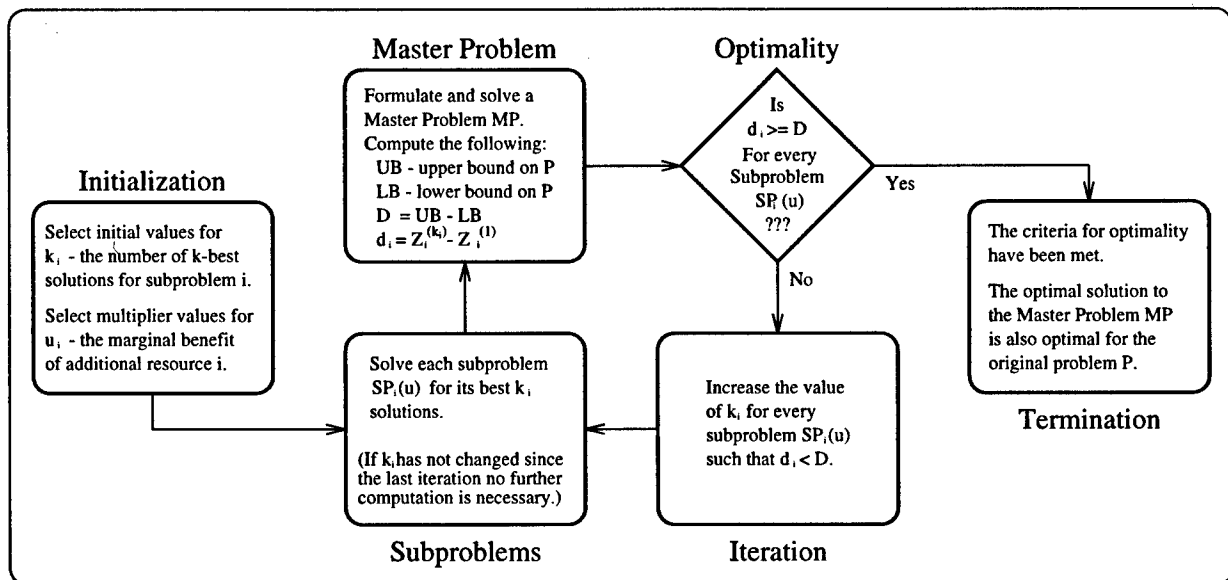


Figure 13 A Flow Diagram of the Sweeney-Murphy Algorithm

**5.1.1 The Sweeney-Murphy Algorithm.** Figure 13 provides a flow diagram to illustrate the basic mechanics of the Sweeney-Murphy integer program decomposition approach while Table 23 offers a detailed description of the Sweeney-Murphy algorithm itself [84]. The Sweeney-Murphy algorithm starts out by assigning initial values for  $k_i$ , the number of  $k$ -best solutions that must be obtained for each subproblem  $i$ , and  $\mu$ , the vector of Lagrangian multipliers required to formulate

the subproblems. Each of the subproblems is then solved for its  $k$ -best solution set, these  $k$ -best solution sets are used to formulate a master problem, and the master problem is solved. Since the master problem, **MP**, is a restriction of the original problem, **P**, any feasible solution to **MP** is also feasible for **P**, although the optimal solution to **MP** is not necessarily optimal for **P**.

Table 23 Sweeney-Murphy Algorithm

- 
- Step 1: (Initialize)  
 Choose an initial  $k_i$  value for each subproblem  $i = 1, \dots, p$ .  
 Find a vector of nonnegative Lagrangian multipliers,  $\mu$ , associated with the  
     Lagrangian relaxation of the coupling constraints in **P**.  
 Use the vector  $\mu$  to formulate the subproblems **SP<sub>i</sub>( $\mu$ )**.  
 Find the top  $k_i$  solutions to each subproblem  $i$ .
- Step 2: (Solve the Master Problem, **MP**)  
 Use the  $k$ -best subproblem solution sets to formulate and then solve **MP**.  
 If no feasible solution exists, repeat the following until one is found:  
     increase  $k_i$  for every subproblem  $i$ ,  
     resolve all of the subproblems for the larger  $k_i$ ,  
     then formulate and solve a new master problem.  
 When a feasible solution is found,  
     let  $\lambda^*$  denote the the optimal solution to **MP**,  
     let  $x_i(\lambda^*)$  be the corresponding solution to **SP<sub>i</sub>( $\mu$ )**,  
     and let  $UB = \sum_{i=1}^p c_i x_i(\lambda^*)$  be the upper bound on the optimal solution to **P**.
- Step 3: (Test For Optimality )  
 Let  $LB(\mu) = \sum_{i=1}^p (c_i - \mu A_i) x_i^1 + \mu b_0$  be a lower bound on the optimal solution to **P**.  
 Set  $\Delta = UB - LB(\mu)$ .  
 Set  $\delta_i = (c_i - \mu A_i) x_i^{k_i} - (c_i - \mu A_i) x_i^1$  for all  $i = 1, \dots, p$ .  
 If  $\delta_i \geq \Delta$  for all  $i = 1, \dots, p$ , go to Step 5.
- Step 4: (Iterate)  
 Increase  $k_i$  for every subproblem  $i$  where  $\delta_i < \Delta$ .  
 If  $\delta_i < \Delta$ , resolve subproblem  $i$  for the larger  $k_i$ .  
 Go to Step 2.
- Step 5: (Terminate)  
 STOP. The current optimal solution to **MP** is optimal for **P**.
- 

Sweeney and Murphy present optimality criteria based on the tightness of the current bounds on the optimal solution to **P** and the range of the objective function values in the  $k$ -best solution sets for the subproblems [84]. Using this information, optimality criteria are established to determine if the current optimal solution to **MP** is also optimal for **P**. If the test fails for any subproblem  $i$ , the corresponding value  $k_i$  must be increased and the subproblem must be re-solved to expand the

$k$ -best solution set accordingly. That is, more options are offered for those subproblems which fail the optimality test. The algorithm continues to iterate between finding larger  $k$ -best solution sets for select subproblems and solving the master problem until the optimality conditions have been met for every subproblem. At this point, the optimal solution to **MP** is also optimal for **P** and the algorithm terminates.

Table 24 Sweeney-Murphy Optimality Theorem

**Theorem** If  $\delta_i \geq \Delta \forall i = 1, \dots, p$ ,  $\{x_1(\lambda^*), x_2(\lambda^*), \dots, x_p(\lambda^*)\}$  is an optimal solution to **P**.

**Proof:**

Define  $\lambda^*$  to be the optimal solution to **MP**,

$x_i(\lambda^*)$  to be the corresponding solution to **SP** <sub>$i$</sub> ( $\mu$ ), and

$\{x_1(\lambda^*), x_2(\lambda^*), \dots, x_p(\lambda^*)\}$  to be the solution to **P** implied by  $\lambda^*$ .

The value of  $\{x_1(\lambda^*), x_2(\lambda^*), \dots, x_p(\lambda^*)\}$  in **P** is an upper bound for **P**.

$$\Rightarrow UB = \sum_{i=1}^p c_i x_i(\lambda^*)$$

Suppose that  $\delta_i \geq \Delta$  for all  $i = 1, \dots, p$ , but  $\{x_1(\lambda^*), x_2(\lambda^*), \dots, x_p(\lambda^*)\}$  is not optimal for **P**.

$$\Rightarrow \exists \bar{j} > k_{\bar{i}} \text{ for subproblem } \bar{i} \text{ such that } x_{\bar{i}}^{\bar{j}} \text{ is part of a feasible solution to } \mathbf{P} \\ \text{with objective function value } \bar{z} < UB.$$

We need to show that this leads to a contradiction.

The minimum value of the Lagrangian relaxation of **P** with  $x_{\bar{i}}$  held constant at  $x_{\bar{i}}^{\bar{j}}$  is given by  $\sum_{i \neq \bar{i}} (c_i - \mu A_i) x_i^1 + (c_{\bar{i}} - \mu A_{\bar{i}}) x_{\bar{i}}^{\bar{j}} + \mu b_0$ .

$$\Rightarrow \bar{z} \geq \sum_{i \neq \bar{i}} (c_i - \mu A_i) x_i^1 + (c_{\bar{i}} - \mu A_{\bar{i}}) x_{\bar{i}}^{\bar{j}} + \mu b_0.$$

Recall that  $LB(\mu) = \sum_{i=1}^p (c_i - \mu A_i) x_i^1 + \mu b_0$  which leads to the relationship

$$\sum_{i \neq \bar{i}} (c_i - \mu A_i) x_i^1 + (c_{\bar{i}} - \mu A_{\bar{i}}) x_{\bar{i}}^{\bar{j}} + \mu b_0 = LB(\mu) + (c_{\bar{i}} - \mu A_{\bar{i}}) x_{\bar{i}}^{\bar{j}} - (c_{\bar{i}} - \mu A_{\bar{i}}) x_{\bar{i}}^1.$$

$$\Rightarrow \bar{z} \geq LB(\mu) + (c_{\bar{i}} - \mu A_{\bar{i}}) x_{\bar{i}}^{\bar{j}} - (c_{\bar{i}} - \mu A_{\bar{i}}) x_{\bar{i}}^1.$$

$$\Rightarrow \bar{z} \geq LB(\mu) + \delta_{\bar{i}}.$$

$$\Rightarrow \bar{z} \geq LB(\mu) + \min_i \{\delta_i\}.$$

$$\Rightarrow \bar{z} \geq LB(\mu) + \Delta.$$

$$\Rightarrow \bar{z} \geq UB.$$

Contradiction! We already assumed that  $\bar{z} < UB$ .

Therefore,  $\{x_1(\lambda^*), x_2(\lambda^*), \dots, x_p(\lambda^*)\}$  is an optimal solution to **P**.

**5.1.2 Sweeney-Murphy Convergence.** Table 24 contains an expanded version of the theorem and proof provided by Sweeney and Murphy to support the convergence criteria used in their integer program decomposition algorithm [84].

The Sweeney-Murphy algorithm is a general approach that is shown to converge for any block-angular integer program. However, there are some aspects of the algorithm for which Sweeney and Murphy offer no specific guidelines: finding  $k$ -best solution sets, choosing initial values for  $k_i$ , choosing new values for  $k_i$ , and finding values for the Lagrangian multipliers. Although the algorithm is theoretically sound, Sweeney-Murphy decomposition has not been widely applied in the literature [20, 84, 85]. In an implementation of the Sweeney-Murphy algorithm, the decisions made regarding these issues can greatly affect the performance of the algorithm. For this reason, there are no guarantees concerning the ease of implementation or the algorithm's efficiency for a given problem type.

In the literature, the few applications of the Sweeney-Murphy algorithm obtained the  $k$ -best subproblem solutions by unsophisticated approaches (repetitively solving, cutting the optimal solution out of the feasible region, and re-solving) and selected initial and subsequent values for  $k_i$  arbitrarily [20, 84, 85]. While Sweeney and Murphy do not clarify these issues, they do, however, offer some advice on how to handle the problem of finding values for the Lagrangian multipliers. They show that the optimal values for the Lagrangian multipliers is given by the optimal solution to **D**, the Lagrangian dual of **P** [84:page 1132].

$$(\mathbf{D}) \quad \max_{\mu \geq 0} LB(\mu) = \max_{\mu \geq 0} \min \left\{ \sum_{i=1}^p (c_i - \mu A_i) x_i + \mu b_0 : B_i x_i \geq b_i \text{ and } x_i \in Z^+ \forall i \right\}$$

In some cases, the Lagrangian dual of **P** might be an easy problem to solve. This would be the case if the subproblem constraint matrices were totally unimodular. For cases where such ideal conditions do not exist, Sweeney and Murphy suggest that the dual variables associated with the linking constraints in an LP relaxation of either **P** or **MP** often provide good values for the Lagrangian multipliers [84:page 1133]. It should be noted, however, that for a given problem type, there are no guarantees of either a well behaved Lagrangian dual or an LP relaxation which yields meaningful dual variables for use as Lagrangian multipliers in the subproblem formulations.

## 5.2 Implementing Sweeney-Murphy Decomposition

Building on the details of the basic Sweeney-Murphy integer program decomposition algorithm, this section focuses on the implementation of such a decomposition approach for the combat planning problem presented in Chapter III. The discussion describes the implementation specific algorithm details that are added to the basic algorithm in order to address the unresolved issues concerning finding  $k$ -best solution sets, choosing initial values for  $k_i$ , choosing new values for  $k_i$ , and finding values for the Lagrangian multipliers. After the details of the implementation are discussed, a simple implicit enumeration approach is presented as a more effective alternative for solving the Sweeney-Murphy master problem.

*5.2.1 Finding the  $k$ -Best Subproblem Solutions.* The problem of enumerating the top  $k$  solutions for every subproblem is a significant hurdle to overcome for any implementation of the Sweeney-Murphy integer program decomposition algorithm. There are several specialized algorithms available in the literature that offer more effective alternatives to brute force for finding the  $k$ -best solutions to certain unique classes of problems [9, 34, 57, 63]. Unfortunately, there is no such alternative for general integer program problems that do not belong to one of these unique problem classes. The few implementations of the Sweeney-Murphy approach found in the literature obtain the  $k$ -best subproblem solutions by repeatedly solving a subproblem and then reformulating the subproblem to exclude the optimal solution found [20, 84, 85]. This approach can be implemented, but it may be so computationally time consuming that it makes the Sweeney-Murphy approach undesirable.

In Chapter IV, a subproblem solution approach was presented that offers both a general approach to finding the  $k$ -best solutions for any integer program and a specific approach that exploits the specific features of the combat planning problem to find the  $k$ -best solutions more effectively. The general approach is to apply a standard branch and bound algorithm for integer programs, but the general approach is modified to maintain a list of the top  $k$  solutions as the solution tree is traversed. This modification affects both the manner in which newly discovered solutions are evaluated and the way the solution tree is pruned.

Until the branch and bound algorithm has encountered its first  $k$  feasible solutions, every new feasible solution discovered is added to a rank ordered solution list. Once this list contains  $k$  entries,

the objective function value of every new feasible solution discovered by the algorithm is evaluated against the objective function value of the current  $k^{th}$  best solution in the rank ordered solution list. If the objective function value of the newly discovered solution is better than that of the current  $k^{th}$  best solution, the current  $k^{th}$  best solution is discarded and the newly discovered solution is added to the rank ordered solution list. Otherwise, the newly discovered solution is discarded.

In a standard integer program branch and bound approach, a branch of the solution tree is pruned when it can be shown that the best possible solution that might be found by exploring that branch is bounded by the current best feasible solution. When the goal is to find the  $k$ -best solutions for an integer program, the standard branch and bound approach may be modified so that branches are pruned based on the bound provided by the current  $k^{th}$  best solution in the rank ordered solution list. This modified pruning strategy ensures that none of the  $k$ -best solutions are overlooked. The theorem and proof in Table 25 show that a standard branch and bound approach modified in this way finds the  $k$ -best solutions for any integer program.

In Chapter IV, a specialized branch and bound procedure was developed for the MMGRCPSP using the modifications described by the theorem in Table 25 to obtain the  $k$ -best active schedules. The combat planning problem formulation proposed in this dissertation is a special case of the MMGRCPSP with a block-angular structure. Since each block in the formulation is a wave of attack missions, the subproblems of a decomposition are temporally discrete. This condition allows for a clean partitioning of the constraints between the subproblems and the master problem. The nonrenewable resource constraints are the linking constraints in the formulation of the Sweeney-Murphy master problem while each renewable resource or precedence constraint becomes a part of the formulation of one of the Sweeney-Murphy subproblems.

It is possible that every schedule in the set of the  $k$ -best active schedules for a given problem instance might have several inactive schedules with the same objective function value. These schedules would represent alternate optimal solutions and are obtained by shifting activity start times within their slack times. Since all of the alternate optimal solutions for a given active schedule have the same nonrenewable resource utilization and contribution to the master problem objective function, there is no need to enumerate the inactive schedules. For this reason, the results

---

Table 25    *K*-Best Solutions Theorem

---

**Theorem**    If a standard integer program branch and bound algorithm is modified to retain a rank ordered list of the top  $k$  feasible solutions encountered and branches are pruned only when dominated by the current  $k^{th}$  best solution, then the resulting algorithm is guaranteed to find the  $k$ -best solutions to any integer program so long as there exist  $k$  feasible solutions to the problem.

**Proof:**

Let  $P$  be any integer program that has at least  $k$  feasible solutions.

Let  $K$  be the set of the  $k$ -best solutions to the integer program  $P$ .

Let  $X = \{x_i : i = 1, \dots, k\}$ , be the rank ordered list of solutions to  $P$  obtained by an integer program branch and bound algorithm modified as described above.

Solutions are ranked by objective function value with  $x_1$  being the best.

Suppose  $\exists x^* \in K$  but  $x^* \notin X$ .     $\Rightarrow$      $z^* < z_k$

This implies that either

- (1)  $x^*$  was encountered, but not added to the rank ordered solution list,
- (2)  $x^*$  was added to the rank ordered solution list and later discarded, or
- (3)  $x^*$  was part of a branch of the solution tree that was pruned.

Case 1: Suppose that  $x^*$  was encountered, but not added to  $X$ .

Let  $\bar{X}$  be the rank ordered solution list when  $x^*$  was encountered.

If  $x^*$  was not added to  $\bar{X}$ , then  $z^* \geq \bar{z}_k$ , but we have already assumed that

$$z^* < z_k \leq \bar{z}_k \quad \Rightarrow \quad \text{Contradiction!}$$

Case 2: Suppose that  $x^*$  was discarded from the rank ordered solution list.

Let  $\bar{X}$  be the rank ordered solution list after  $x^*$  was discarded.

If  $x^*$  was discarded, then  $z^* > \bar{z}_k$ , but we have already assumed that

$$z^* < z_k \leq \bar{z}_k \quad \Rightarrow \quad \text{Contradiction!}$$

Case 3: Suppose that  $x^*$  was part of a branch pruned from the solution tree.

Let  $\bar{X}$  be the rank ordered solution list at the time the branch of the solution tree containing  $x^*$  is pruned.

Without loss of generality, let  $x^*$  be the best solution on the branch pruned.

If the branch was pruned, then  $z^* > \bar{z}_k$ , but we have already assumed that

$$z^* < z_k \leq \bar{z}_k \quad \Rightarrow \quad \text{Contradiction!}$$

Since all three cases lead to contradiction,  $x^*$  cannot exist. Therefore,  $X = K$ .

---



from the extended Sprecher subproblem solution algorithm may be directly used for solving the Sweeney-Murphy subproblems,  $SP_i(\mu)$ .

If the combat planning problem were to be decomposed based on some criteria other than waves of attack missions, the condition of temporally discrete subproblems might not exist, and the alternate optimal solutions would have to be enumerated. It would still be useful to enumerate the  $k$ -best active schedules with the extended Sprecher algorithm, but it would be necessary to enumerate the inactive schedules associated with those active schedules. Every active schedule has an associated set of activity mode assignments. For a given set of mode assignments, standard CPM techniques may be applied to obtain the earliest and latest possible start times for every activity.

The active schedule is the schedule with every activity beginning execution at its earliest possible start time, while every other feasible combination of activity start times represents an inactive schedule and an alternate optimal solution. It is a relatively simple, but potentially time consuming, procedure to enumerate every feasible combination of start times for a given set of mode assignments to develop the set of inactive schedules associated with an active schedule. However, there is a possibility of large numbers of alternate optimal solutions which may lead to correspondingly large values for  $k_i$  before the Sweeney-Murphy optimality criteria are satisfied. The consequences of these large values for  $k_i$  are addressed in the following section.

*5.2.2 Setting and Resetting the Value of  $k_i$ .* The best way to set  $k_i$ , the number of  $k$ -best solutions obtained for subproblem  $i$ , at each iteration of the Sweeney-Murphy algorithm is an issue for which Sweeney and Murphy offer no guidance. In the few applications of the Sweeney-Murphy algorithm found in the literature, the values for  $k_i$  are chosen arbitrarily, with no apparent theoretical justification [20, 84, 85]. This section presents a discussion of the complexity of the Sweeney-Murphy algorithm in order to evaluate the merits of potential  $k_i$  selection strategies. The affects of the  $k_i$  values on the complexity of the Sweeney-Murphy algorithm are examined in terms of the size of the solution trees, the effort required at each node within the solution trees, and the number of nodes that are visited.

In the extended Sprecher subproblem solution algorithm, the value of  $k_i$  affects neither the size of the complete solution tree to be implicitly enumerated nor the amount of work to be accomplished at each node. Instead, the value of  $k_i$  affects the number of nodes that are visited before the algorithm

converges. To understand why this is the case, consider the total execution time of the extended Sprecher algorithm as two distinct phases, a ramp-up phase and a pruning phase.

The ramp-up phase is the portion of the algorithm's total execution time from the time when the algorithm begins execution until the time when the  $k_i^{th}$  feasible solution is encountered. Since the ramp-up phase lasts until  $k$  feasible solutions have been encountered, it is obvious that larger values for  $k_i$  lead to a correspondingly longer ramp-up phase.

The pruning phase is the portion of the algorithm's total execution time after the  $k_i^{th}$  feasible solution has been found. In the pruning phase, the algorithm prunes bounded branches. Branches are pruned based on an upper bound provided by the objective function value of the current  $k_i^{th}$  best solution. A higher value for  $k_i$  leads to a correspondingly higher value for the objective function of the  $k_i^{th}$  best solution and a looser upper bound. This would indicate that higher  $k_i$  values mean weaker upper bounds, less opportunity for pruning, and a longer pruning phase.

Clearly, while the value of  $k_i$  does not affect the size of the solution tree or the amount of work accomplished at each node, larger  $k_i$  values do require visiting more nodes in order to implicitly enumerate all potential solutions (the size of the tree that is *implicitly* enumerated does not change, but more of its nodes are *explicitly* enumerated). When the algorithm is required to visit more nodes, solution time naturally increases, but to acquire a better understanding of the magnitude of the solution time increase some experiments were performed using the two sets of test problems from Chapter IV. Recall that each test set contains 180 test problems spanning a wide range of possible values for the complexity parameters. The first set was constructed to be comparable to Sprecher's test problems while the second set was constructed to be more representative of actual combat planning problem instances [79].

All 180 problems from each test set were solved for various values of  $k$  using the extended Sprecher subproblem solution algorithm on a 167MHz UltraSPARC with 128 Mb of memory. The test problems were first solved for  $k = 1$  and then solved again with the value of  $k$  ranging from 10 to 100 ( $k = 10, 20, \dots, 100$ ). An average solution time value was computed for each test set at each level of  $k$ . The average solution time data is reported on the graph in Figure 14.

The empirical results in Figure 14 suggest that the relationship between the value of  $k_i$  and the solution time for subproblem  $i$  is at worst linear, at least for the range of problems represented

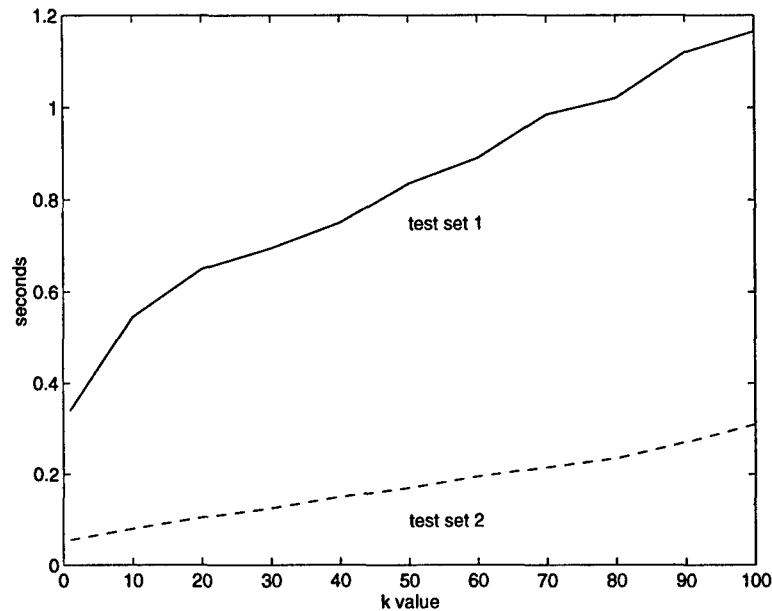


Figure 14 Empirical Results for  $k$ -Best Complexity

in these two test sets. In Chapter IV, it was shown that subproblem solution time is exponentially related to the number of activities in the subproblem. The linear affect of  $k_i$  on subproblem solution time is relatively minor when compared to the exponential affect of the number of activities.

Another Sweeney-Murphy complexity issue that must be investigated is the affect of  $k_i$  on the master problem solution time. For ease of illustration, it is assumed that  $k_i = k$  for every subproblem  $i$ . It is also assumed that the master problem is solved with a standard branch and bound integer program solver that can make use of special ordered sets to reduce the size of the solution tree. Unlike the subproblems, the  $k_i$  value affects both the size of the solution tree and the amount of work that must be accomplished at each node for the master problem.

Suppose that for a given problem, there are  $p$  distinct subproblems. If each subproblem is solved for the top  $k$  solutions, then the formulation of the Sweeney-Murphy master problem has  $p$  special ordered sets of  $k$  binary variables each. Without the special ordered sets, the solution tree for the master problem would contain  $2^{kp}$  possible solutions indicating an exponential relationship between the value of  $k$  and the size of the solution tree. Exploiting the special ordered sets reduces the total number of possible solutions to  $k^p$  and makes the relationship between  $k$  and the solution tree size polynomial.

In Chapter IV, it was shown that a lower bound on the worst case computational effort required at each node for a standard integer program branch and bound solver can be expressed by  $(n/m)^m$ , where  $n$  is the number of variables and  $m$  is the number of constraints. Since  $n = kp$  in the master problem, this upper bound may be reformulated as  $k^m(p/m)^m$  indicating that there is a polynomial relationship between  $k$  and the worst case effort required at each node for the master problem.

While  $k$  affects the size of the master problem solution tree and the amount of effort required at each node within that tree,  $k$  has no affect on what portion of the total number of nodes are visited before a standard integer program branch and bound solver converges. Table 26 summarizes the affects of  $k$  on the complexity of the complete Sweeney-Murphy decomposition algorithm.

Table 26 How  $k$  Affects Sweeney-Murphy Complexity

	Tree Size	Node Effort	Nodes Visited
Subproblems	N/A	N/A	linear
Master Problems	polynomial	polynomial	N/A

When implementing a strategy for setting and resetting the values of  $k_i$  in an implementation of the Sweeney-Murphy decomposition algorithm, the primary concern is how this strategy effects solution time. Based on the complexity information summarized in Table 26, the values for  $k_i$  have a greater affect on the master problem solution time than they do on the subproblem solution times. The overall solution time for the decomposition algorithm may be divided into three parts; the time spent obtaining Lagrangian multipliers, the time spent solving the subproblems, and the time spent solving the master problem.

If the subproblems are *small*, the master problem solution algorithm accounts for the largest portion of the overall solution time and the  $k_i$  selection strategy should be chosen accordingly. If the subproblems are *large*, the subproblem solution algorithm requires the largest portion of the overall solution time and this should be taken into account when the  $k_i$  selection strategy is chosen. Empirical testing is conducted in Appendix A in order to determine the breakpoint, in terms of number of activities, between *small* and *large* subproblems. Based on the results in Appendix A, the strategy chosen for the Sweeney-Murphy implementation in this dissertation is to start with a small initial  $k_i$  value ( $k_i = 10$ ), double  $k_i$  for early iterations until  $k_i$  exceeds 100, and then add 50 to  $k_i$  for every additional iteration.

It is possible, in problem instances with abundant nonrenewable resources, that the master problem can achieve optimality with very small  $k_i$  values. The justification for starting with small  $k_i$  values is that the initial iteration is computationally inexpensive and problems with abundant resources can be solved in a single iteration. When resources are not abundant, large  $k_i$  values are often required before the master problem is optimal. This observation is supported by experience with the test problems used in this dissertation. This experience indicates that there are typically many alternate optimal solutions in problems of this type (MMGRCPSP). The rationale behind doubling  $k_i$  for early iterations is to raise  $k_i$  quickly while iterations are still computationally inexpensive. Once  $k_i$  passes 100, the increase in iteration times becomes significant and doubling  $k_i$  is too costly. For this reason,  $k_i$  is increased by a constant amount rather than doubling in later iterations.

Different  $k_i$  selection strategies were proposed and evaluated based on the criteria suggested by a complexity analysis of the Sweeney-Murphy algorithm. The eventual choice of a strategy to implement was based on trial-and-error and empirical experiments (see Appendix A). It should be noted that the best  $k_i$  selection strategy depends on the type of problems that are to be solved. There is no  $k_i$  selection strategy that is best for all problem types, but a good general rule is to choose a strategy based on the size and complexity of the subproblems. If the subproblems are computationally tasking, the  $k_i$  values should be increased at a quicker rate to limit the total number of Sweeney-Murphy iterations. If the subproblem solution times are small, large numbers of iterations are less of a problem and the  $k_i$  values may be increased at a slower rate to limit the size of the master problems since their size is polynomially related to  $k_i$ .

**5.2.3 Finding Lagrangian Multipliers.** The Sweeney-Murphy integer program decomposition algorithm converges regardless of what values are used for the Lagrangian multipliers in the subproblem objective functions. The problem is that if *bad* values are chosen, the  $k_i$  values might become excessively large before the algorithm achieves optimality. It has already been shown that large  $k_i$  values lead to long subproblem and master problem solution times so it is important to choose *good* values for the Lagrangian multipliers. In fact, choosing the best possible values for Lagrangian multipliers is a critical factor in minimizing solution time for a Sweeney-Murphy implementation. Unfortunately, it is not always easy to obtain *good* Lagrangian multipliers.

Sweeney and Murphy point out that the optimal value for Lagrangian multipliers is given by the solution to the Lagrangian dual of **P**. The subproblem constraint matrices for the combat planning problem are not totally unimodular so it is not practical to solve the Lagrangian dual for the application in this dissertation. Sweeney and Murphy suggest that an acceptable alternative is to use the dual variables associated with the linking constraints in an LP relaxation of either **P** or **MP**. Because of the generalized precedence constraints in the combat planning problem formulation, dual variables from an LP relaxation would not provide meaningful dual variables.

Recall that every pair of activities  $(i, j)$  with a generalized precedence requirement has a different precedence constraint for every possible combination of modes  $(m_i, m_j)$ . Depending on the execution modes selected, only one of these constraints is enforced; the rest are relaxed. This selective constraint enforcement is accomplished using the binary decision variables already present in the model formulation. In an LP relaxation, those decision variables are no longer capable of enforcing any of the precedence constraints. Since the relaxed model has no precedence enforced, its dual variables are not meaningful to the original problem.

For the combat planning model in this dissertation, it is not possible to use either the optimal or the alternate methods suggested by Sweeney and Murphy for selecting Lagrangian multipliers. It is necessary to find another method for obtaining Lagrangian multipliers that are *meaningful* in order to avoid the large  $k_i$  values associated with using poor Lagrangian multipliers. The alternate method proposed in this section parametrically varies resource availability in the subproblems to obtain Lagrange multipliers that have an interpretation similar to that of dual variables for a linear program.

Consider the purpose the Lagrangian multipliers serve in the objective functions of the Sweeney-Murphy subproblems. Setting these multipliers to zero allows the subproblems to be optimized without any consideration for the utilization of the limited nonrenewable resources. Each subproblem uses these nonrenewable resources without direct consideration of availability or the resource needs of the other subproblems. The purpose of positive-valued Lagrangian multipliers in the subproblem objective functions is to convey how critical each resource is to the overall problem. With meaningful positive-valued multipliers, the subproblem solutions are steered away from unnecessary consumption of the more critical resources.

In an LP relaxation of  $P$ , the dual variable associated with a nonrenewable resource constraint provides an estimate of how critical that resource is to the problem in question. If a resource is scarce, and highly demanded by the subproblems, the dual variable for that resource would be higher than dual variables for less critical resources. The strict interpretation of the value of a dual variable is the marginal benefit of an additional unit of resource at optimality. For this Sweeney-Murphy implementation, the goal is to compute values that, like LP dual variables, represent the marginal benefit of an additional unit of each nonrenewable resource.

Nauss proposes that by varying the right-hand-side levels one at a time, re-solving the integer program problem, and observing the change in the objective function, it is possible to estimate the marginal benefit of an additional unit of resource [64]. Values for marginal benefit obtained in this manner would provide appropriate Lagrangian multipliers, but there is a problem with this approach. Nauss' approach requires that the Sweeney-Murphy master problem be solved iteratively, varying the right-hand-side level for each nonrenewable resource in order to compute values for marginal benefit to use as Lagrangian multipliers. Unfortunately, the  $k$ -best subproblem solutions are necessary to formulate the master problem and the Lagrangian multipliers are necessary to formulate the subproblems, so until Lagrangian multipliers are found, there is no master problem to solve. The procedure described in Table 27 is a variation on the parametric analysis approach proposed by Nauss.

The procedure described in Table 27 uses parametric analysis to find the marginal benefit of each nonrenewable resource in the Sweeney-Murphy master problem. These marginal benefit values have the same interpretation as dual variables in a linear program—the amount of objective function improvement per unit increase in resource availability. Marginal benefit is a good estimate of the value of each resource to the master problem. These values provide a reasonable set of Lagrangian multipliers for formulating the Sweeney-Murphy subproblems.

One aspect of the procedure for calculating marginal benefit is not included in the description in Table 27. That aspect is the method for finding a feasible partitioning of the nonrenewable resources between the subproblems. This initial allocation of resources could be obtained by applying any heuristic method to find a feasible solution to  $P$ . The resource utilization in this feasible solution provides a feasible allocation of resources. For the Sweeney-Murphy implementation, the extended

Table 27 Finding Lagrangian Multipliers

---

Step 1: (Initialize)	<p>Set <math>R</math> to the total number of nonrenewable resources in the problem.</p> <p>Find a feasible partitioning of these resources between the subproblems.</p> <p>Solve the subproblems using this feasible resource allocation.</p> <p>The optimal subproblem solutions provide a feasible solution to the master problem.</p> <p>Set <math>Z^*</math> to the master problem objective function value for this feasible solution.</p> <p>Set <math>\bar{U}_r</math> to the resulting utilization of each resource <math>r = 1, \dots, R</math>.</p>
Step 2: (Relax the Subproblem Resource Constraints)	<p>For <math>r = 1, \dots, R</math> do the following.</p> <p>Solve the subproblems with the same feasible allocation of resources, but relax the constraint associated with nonrenewable resource <math>r</math>.</p> <p>Set <math>Z_r</math> to the resulting master problem objective function value.</p> <p>Set <math>U_r</math> to the resulting utilization of each resource <math>r = 1, \dots, R</math>.</p>
Step 3: (Set Lagrangian Multipliers to the Marginal Benefit)	<p>For <math>r = 1, \dots, R</math> do the following.</p> <p>If <math>(U_r - \bar{U}_r) &gt; 0</math>, then set <math>\mu_r = (Z^* - Z_r)/(U_r - \bar{U}_r)</math>.</p> <p>Else, set <math>\mu_r = 0</math>.</p>

---

Sprecher algorithm was modified to terminate after one feasible solution is found. This feasible solution is then used to obtain the feasible partitioning of resources between the subproblems necessary for calculating the marginal benefit of each nonrenewable resource.

A limitation of the marginal benefit approach described in Table 27 is that the quality of the initial feasible solution used to obtain an initial partitioning of resources may have a direct affect on the quality of the Lagrangian multipliers derived from the resulting marginal benefit values. Dual variables are a measure of the marginal benefit of a resource at the optimal solution of an LP. The *proxy dual variables* of Table 27 are a measure of the marginal benefit of a resource at some feasible solution of an IP. The *proxy dual variables* are less reliable when the initial feasible solution is farther from the optimal solution. Poor Lagrangian multiplier values may lead to large  $k_i$  values and large solution times.

To account for the possibility of poor initial Lagrangian multiplier values, the Sweeney-Murphy implementation in this dissertation adjusts the multiplier values as soon as a feasible solution is found for the master problem. Once a feasible master problem has been constructed and solved, the same parametric analysis shown in Table 27 is applied to the master problem to obtain revised estimates



of the marginal benefit of each nonrenewable resource. This is accomplished by iteratively re-solving the master problem with one resource constraint relaxed. Each time a relaxed master problem is solved, the differences between the original and relaxed objective function values and the original and relaxed resource utilization are used to compute marginal benefit for the relaxed resource. These new marginal benefit values replace the initial marginal benefit values as the Lagrangian multipliers used in the formulation of the subproblems.

*5.2.4 Solving the Sweeney-Murphy Master Problem.* The Sweeney-Murphy master problem is an integer program consisting of all of the linking constraints, an additional constraint to deal with solution selection for each subproblem, and  $k_1 + k_2 + \dots + k_p$  binary variables. This problem may be solved with any standard integer program solver, and if the solver takes advantage of special ordered sets, solution time may be improved. However, as the  $k_i$  values become larger in later Sweeney-Murphy iterations, solution of the master problem may become computationally daunting. For this reason, a specialized master problem solution algorithm is proposed to solve master problem instances faster than would be possible with a standard integer program solver. The proposed algorithm also includes the capability to obtain  $k$ -best solution sets for the master problem. When the optimality criteria have been met, the  $k$ -best solutions to the master problem correspond to the  $k$ -best solutions to the original problem and present the decision maker with a set of alternate optimal or near-optimal solution alternatives to consider.

The specialized master problem solution algorithm is a simple branch and bound approach that implicitly enumerates the same solution tree searched by a standard integer program solver. The solution tree includes every possible combination of subproblem solutions. A standard integer program solver would search this tree by solving an LP relaxation at each node and pruning branches based on the objective function value of the best integer solution found. The specialized algorithm searches the same tree, pruning branches based on simple bounding or resource limitations, but it only requires feasibility checks at each node. The feasibility checks in the specialized algorithm require substantially less computational effort than the solution of the LP relaxations in a standard solver. The reduced computational effort required at each node in the solution tree allows the specialized algorithm to solve master problem instances faster than a standard integer program

solver and provides the additional benefit of  $k$ -best solution sets. The specialized master problem solution algorithm is given in Table 28.

The solution tree traversed by the specialized master problem solution algorithm described in Table 28 has  $p$  levels, one for each subproblem  $i = 1, \dots, p$ . As the solution tree is traversed, a partial solution is maintained by assigning subproblem solutions as the algorithm moves deeper into the tree and unassigning subproblem solutions when the algorithm backtracks. At level  $i$  of the solution tree, the algorithm assigns one of the  $k_i$  solutions for the subproblem  $i$  and then attempts to prune the current branch. The current branch may be pruned if the best possible completion to the current partial solution is dominated by the current  $k^{th}$  best master problem solution. Pruning is also possible if, for any resource  $r$ , there is not enough available resource remaining to feasibly complete the current partial solution.

When either of the pruning criteria are met, all completions of the current partial schedule are eliminated from consideration and the algorithm backtracks up one level in the solution tree. If neither pruning criteria is met, the algorithm moves a level deeper into the solution tree and assigns a solution for the next subproblem. When the algorithm reaches the bottom of the solution tree and a solution has been assigned for subproblem  $p$ , a feasible master problem solution has been found. At this point, the solution is added to the  $k$ -best solution list and the algorithm backtracks one level. When the entire solution tree has been implicitly enumerated, the algorithm terminates.

### 5.3 Testing the Sweeney-Murphy Implementation

This section reports the results of testing conducted on the implementation of the Sweeney-Murphy integer program decomposition algorithm described in this chapter. While the subproblem solution algorithm testing focused on the affects of complexity parameters on solution time, the decomposition algorithm testing holds the complexity parameters constant and studies the relationship between size parameters and solution time. The two size parameters that are investigated are  $A$ , the number of activities per subproblem, and  $P$ , the number of subproblems per problem instance. These two parameters determine the total number of activities in each problem instance and are a good indicator of the size of the overall problem.

Table 28 Master Problem Solution Algorithm

---

Step 1:	(Initialize)
	Set $P$ to the total number of projects.
	Set $R$ to the total number of resources.
	Set $A_r$ to the total available quantity of resource $r$ .
	Set $K$ to the number of $k$ -best master problem solutions to find.
	Let $p$ be the current subproblem and set $p = 1$ .
	Let $s_p$ be the solution selected for subproblem $p$ and set $s_1 = 0$ .
	Set $z_{sp}$ to the contribution to the master problem objective function for the $s^{th}$ solution to subproblem $p \quad \forall p = 1, \dots, P$ and $s = 1, \dots, k_p$ .
	Let $Z_p$ be the sum of the contributions to the master problem objective function for subproblems $1, \dots, p$ and set $Z_0 = 0$ .
	Set $\bar{Z}_p = \sum_{i=p+1}^P \min_{1 \leq s \leq k_i} (z_{si}) \quad \forall p = 1, \dots, (P-1)$ .
	Set $u_{rsp}$ to the quantity of resource $r$ used by the $s^{th}$ solution to subproblem $p \quad \forall r = 1, \dots, R, p = 1, \dots, P$ , and $s = 1, \dots, k_p$ .
	Let $U_{rp}$ be the combined utilization of resource $r$ for subproblems $1, \dots, p$ and set $U_{r0} = 0 \quad \forall r = 1, \dots, R$ .
	Set $\bar{U}_{rp} = \sum_{i=p+1}^P \min_{1 \leq s \leq k_i} (u_{rsi}) \quad \forall r = 1, \dots, R$ and $p = 1, \dots, (P-1)$ .
	Let $ub$ be the upper bound on the optimal solution to MP and set $ub$ to some arbitrarily large number, $M$ ( $ub = M$ ).
Step 2:	(Select the Next Subproblem Solution)
	Set $s_p = s_p + 1$ .
	If $s_p \leq k_p$ ,
	Set $Z_p = Z_{p-1} + z_{s_p p}$ .
	Set $U_{rp} = U_{r(p-1)} + u_{rs_p p} \quad \forall r = 1, \dots, R$ .
	Go to Step 4.
Step 3:	(Backtrack)
	Set $p = p - 1$ .
	If $p < 1$ , STOP!!! The $k$ -best solutions for MP have been found.
	Else, Go to Step 2.
Step 4:	(Try to Prune)
	If $((Z_p + \bar{Z}_p) > ub)$ or $((U_{rp} + \bar{U}_{rp}) > A_r$ for any $r$ ), Go to Step 2.
Step 5:	(Move to the Next Project)
	Set $p = p + 1$ .
	If $p \leq P$ , then set $s_p = 0$ and Go to Step 2.
	Else, set $p = p - 1$ and Go to Step 6.
Step 6:	(Store the Current Solution)
	Add solution $(s_1, s_2, \dots, s_P)$ to the list of master problem solutions.
	If the list contains more than $K$ solutions, discard the $(K + 1)^{th}$ best solution and set $ub$ to the objective function value of the $K^{th}$ best solution.
	Go to Step 2.

---

The relationship between the size parameters  $A$  and  $P$  and problem solution time is examined by applying the Sweeney-Murphy implementation to a number of test problems that cover a range of size parameter values. To standardize the test results reported in this dissertation, the Sweeney-Murphy implementation is tested under the same conditions as the subproblem solution algorithm. The Sweeney-Murphy implementation is coded in C and uses the standard C function *gettimeofday* to calculate and report solution times in seconds. The test runs are performed on a 486DX66 processor and solution times are reported to the nearest tenth of a second.

When the subproblem solution algorithm was tested, two different sets of test problems were used. The first set was designed to be similar to the test problems used by Sprecher while the second set was designed to have properties that would be expected in combat planning problem instances. This section makes no direct comparisons with Sprecher's work so there is no reason for the test problems to look like Sprecher's problems. Instead, the test problems are generated to be more like combat planning problems using the ProGen parameter values given in Chapter IV. Additionally, the parameters for resource strength are set to the values that presented the most complex problem instances as indicated by the testing results in Chapter IV. Table 29 provides a summary of the fixed ProGen parameters that are used to generate the test problems for the Sweeney-Murphy implementation.

Table 29 Fixed Parameters for the Sweeney-Murphy Test Problems

	$ M_i $	$\tau_{im}$	$ K $	$U_K$	$Q_K$	$ S_1 $	$ S_j $	$ P_j $	$ P_j $	$PRS$	$TRS$
min	1	1	3	2	1	3	1	3	1	0.5	0.5
max	3	10	3	4	1	3	3	3	3	0.5	0.5

In Table 29, the values for the parameters  $|M_i|$  and  $\tau_{im}$  indicate that each activity may have between one and three possible execution modes and the duration associated with any mode is restricted to be between one and ten time units. While the duration restriction is arbitrary, the restriction on the number of execution modes is intended to simulate the limited number of weaponeering options available for each target on the target nomination list. The values for  $|K|$ ,  $U_K$ , and  $Q_K$  specify that every problem instance has exactly three doubly constrained resources and every activity execution mode requires between two and four units of exactly one resource. Again, the number of resources is arbitrary, but the other information is intended to simulate a weaponeering

option that calls for two to four aircraft of the same configuration to be tasked against a specific target.

The parameters  $|S_1|$ ,  $|S_j|$ ,  $|P_J|$ , and  $|P_j|$  govern the construction of the precedence network for each problem instance. The values of these parameters cause the origin activity to have exactly three successors, cause the terminal activity to have exactly three predecessors, and restrict all other activities to between one and three predecessors and between one and three successors. The last two fixed parameters,  $PRS$  and  $TRS$ , are measures of resource strength and specify the level of resource availability in the test problems. These parameters may be assigned any value between zero and one where higher values indicate more abundant resources. The tests in Chapter IV indicate that problems with scarce resources require longer solution times. To make the test problems for the Sweeney-Murphy implementation as challenging as possible, the values for the resource strength parameters were set at the lowest level tested in Chapter IV.

The two ProGen parameters that are varied are  $A$ , the number of activities in each subproblem, and  $P$ , the number of subproblems in each problem instance. Each of these parameters is investigated at three different levels: the levels of  $A$  are 10, 15, and 20 and the levels of  $P$  are 2, 3, and 4. ProGen is used to generate ten problem instances for every combination of the levels of these parameters resulting in a total of 90 test problems. The Sweeney-Murphy implementation is applied to all 90 problems and the results, reported in seconds on a 486DX66 processor, are shown in Table 30.

Table 30 Sweeney-Murphy Test Results

		P=2			P=3			P=4			Total
		min	avg	max	min	avg	max	min	avg	max	avg
A=10	k	5	31.70	180	5	6.67	40	5	7.50	40	15.29
	i	0	4.2	10	0	2.2	6	0	2.2	7	2.9
	t	0.1	1.3	2.6	0.1	1.0	1.5	0.2	1.3	3.5	1.2
A=15	k	5	121.00	2080	5	140.00	1480	5	14.12	80	91.71
	i	0	7.7	47	0	7.9	34	0	2.8	7	6.1
	t	0.3	84.3	638.5	0.2	83.6	471.7	1.2	17.0	41.1	61.6
A=20	k	5	44.00	480	5	41.33	1030	5	8.00	20	31.11
	i	0	3.9	15	0	4.9	25	0	1.5	3	3.4
	t	2.0	73.8	284.1	15.0	147.1	680.1	8.9	79.7	264.1	100.2
Total	k		65.57			62.67			9.87		46.04
	i		5.3			5.0			2.2		4.2
	t		53.1			77.2			32.7		54.3

The goal of this test is to explore the relationship between the size parameters and the problem solution times. However, the solution time results are difficult to interpret alone. For this reason, Table 30 reports information about the size of the  $k$ -best subproblem solution sets ( $k$ ) and the number of Sweeney-Murphy iterations ( $i$ ) as well as solution time ( $t$ ). Note that in the process of finding initial values for the Lagrangian multipliers, the subproblems are solved separately with the linking constraints relaxed. If the combination of optimal relaxed subproblem solutions is feasible with respect to the linking constraints, that combination of partial solutions is optimal to the original problem. In such cases, the number of iterations is recorded as 0 because the Sweeney-Murphy master problem was never solved. In all other cases, the value recorded for  $i$  corresponds to the number of times the Sweeney-Murphy master problem is solved.

The aggregate average solution times given in the last column of Table 30 behave as expected. Solution times grow as the number of activities per subproblem grows, but beyond this one observation, the solution times appear somewhat erratic. This erratic behavior has two causes. The first cause is the influence of the size of the  $k$ -best subproblem solution sets and the number of Sweeney-Murphy iterations. The second cause is the affect of *outlier* observations. Consider the information in Table 30 pertaining to the case where  $A = 15$  and  $P = 2$ . If the observation with the largest solution time is excluded, the average solution time drops from 84.3 to 22.7. Obviously, the algorithm is susceptible to certain problem instances that require considerably more time and iterations to achieve optimality. These *outlier* problem instances have excessive alternate optimal subproblem solutions that lead to high numbers of Sweeney-Murphy iterations and the result is the wide range of observations in the test results.

Even with the wide range of observations caused by the outlier observations, the test results can be reorganized to make the affect of the size parameters on solution time clearer. In Table 31, the rows represent different levels of the total number of problem activities (TA) instead of the number of activities per subproblem. In this way, it is possible to investigate the relationship between problems that have the same total number of activities, but different numbers of subproblems. It is expected that if two problem instances have the same total number of activities, the one that can be partitioned into more, smaller subproblems should be easier to solve.

Table 31 Sweeney-Murphy Test Results Reorganized

		P=2			P=3			P=4			Total
		min	avg	max	min	avg	max	min	avg	max	avg
TA=30	k	5	121.00	2080	5	6.67	40	N/A			63.84
	i	0	7.7	47	0	2.2	6				5.0
	t	0.3	84.3	638.5	0.1	1.0	1.5				42.7
TA=60	k	N/A			5	41.33	1030	5	14.12	80	27.73
	i				0	4.9	25	0	2.8	7	3.9
	t				15.0	147.1	680.1	1.2	17.0	41.1	82.1
Total	k	121.00			24.00			14.12			45.78
	i	7.7			3.6			2.8			4.4
	t	84.3			74.1			17.0			62.4

The information in Table 31 is still influenced by the extreme outlier observations, but it is possible to make a few meaningful observations. The first two observations are exactly as would be expected. First, if one problem instance has significantly more total activities than another, it takes longer to solve. Second, if two problem instances have the same total number of activities, the one with fewer subproblems takes longer to solve. The final observation is less obvious, but helps to explain an apparent paradox observed in the test results.

Consider the row in Table 30 where  $A = 15$ . The first entry in this row corresponds to problems with 30 total activities, the second to problems with 45 total activities, and the third to problems with 60 total activities. Normally, the expected solution time results would be increasing with the increasing number of total activities, but the observed results are the exact opposite, hence the apparent paradox. Aside from outlier behavior, the paradox is explained by a general tendency for problem instances with more subproblems to require less Sweeney-Murphy iterations and smaller values for  $k_i$ .

This observation is not as obvious as the first two, but makes sense when the task of the Sweeney-Murphy master problem is considered. The task of the master problem is to select the best combination of subproblem solutions. Consider two problem instances, each with an unknown number of total activities. Suppose problem  $A$  has four subproblems while problem  $B$  has only two. Now, let  $k_i = 10$  for every subproblem of both problem instances. The formulation of the master problem for each problem instance is independent of the number of total activities in the problem instance. The master problem for  $A$  has  $10^4 = 10000$  possible combinations of subproblem solutions

to choose from while the master problem for  $B$  has only  $10^2 = 100$ . Regardless of the number of total activities, the master problem for  $A$  has more possible solutions and potentially a greater chance to achieve optimality at lower values of  $k_i$ .

With this in mind, it becomes clearer why problem instances with more subproblems may be solved with smaller values for  $k_i$  and hence fewer Sweeney-Murphy iterations. Fewer iterations generally mean smaller solution times and this helps to explain the confusing results where, in some cases, smaller problems take longer to solve than larger ones. In addition to explaining the solution time paradox, this algorithm behavior offers the hope of small numbers of iterations for large problems as long as the problems can be partitioned into a sufficient number of subproblems. Small numbers of iterations could perhaps offset the large subproblem solution times that are required for larger problem instances.

To reinforce the observations made thus far, a new set of test problems is constructed according to the reorganization applied in Table 31. The performance of the Sweeney-Murphy algorithm on this new set of test problems, reported in seconds on a 486DX66 processor, is summarized in Table 32.

Table 32 More Sweeney-Murphy Test Results

		P=2			P=3			P=4			Total
		min	avg	max	min	avg	max	min	avg	max	avg
TA=24	k	5	8.50	20	5	5.17	10	5	3.45	10	5.71
	i	0	2.9	6	0	2.2	4	0	1.8	3	2.3
	t	0.1	2.0	4.6	0.0	0.6	1.3	0.0	0.5	2.0	1.0
TA=36	k	5	44.00	330	5	12.41	80	5	7.13	40	21.18
	i	0	4.0	11	0	2.7	6	0	2.4	5	3.0
	t	2.4	63.6	173.6	0.1	2.0	3.6	0.1	0.9	1.8	22.2
TA=48	k	5	76.30	730	5	45.17	230	5	6.88	40	42.78
	i	0	9.2	40	0	6.8	49	0	2.0	6	6.0
	t	12.4	1074.9	5444.4	0.5	153.5	1382.8	0.1	2.2	5.7	410.2
Total	k		42.93			20.92			5.82		23.22
	i		5.4			3.9			2.1		3.8
	t		380.2			52.0			1.2		144.5

The results in Table 32, while still influenced by outliers, clearly exhibit the trends observed previously. In the table, each row reports the results for problems with the same total number of activities while each column reports the results for problems with the same number of subproblems.



Table 32 reports three measures of performance, the average size of the  $k$ -best subproblem solution sets when the algorithm achieves optimality ( $k$ ), the average number of Sweeney-Murphy iterations required to achieve optimality ( $i$ ), and the average computation time required to achieve optimality ( $t$ ). For these measures of performance, smaller values are more desirable, and the results in Table 32 show that each of these measures of performance increases as the total number of activities increases or as the number of subproblems decreases. The general observation is not surprising. Problem instances with more total activities and fewer subproblems are the most difficult problems to solve.

#### 5.4 Summary

This chapter implemented a Sweeney-Murphy integer program decomposition approach for large MMGRCPSP instances with block-angular structure. In doing so, it was necessary to overcome the three troublesome aspects of Sweeney-Murphy decomposition. The  $k$ -best solutions feature of the subproblem solution algorithm from Chapter IV accounted for the first aspect. Parametric analysis was applied to satisfy the requirement for Lagrangian multipliers and complexity analysis was used to determine an effective strategy for setting and resetting the  $k_i$  values. In Chapter VI, an evolutionary algorithm approach to the MMGRCPSP is developed to provide a heuristic solution alternative. The evolutionary algorithm in Chapter VI is combined with the decomposition method from this chapter to form a hybrid decomposition approach that offers a reduction in solution times for larger problem instances.

## VI. *An Evolutionary Algorithm Approach*

The combination of the implicit enumeration algorithm developed in Chapter IV with the decomposition approach developed in Chapter V provides a means for optimizing large MMGRCPSP instances more effectively than any other method currently available. Although the methods presented in Chapters IV and V significantly increase the size of MMGRCPSP instances that may be optimized subject to realistic time limitations, the MMGRCPSP is NP-Complete and the order of the complete decomposition approach remains exponential. The limitations of computer hardware and available computation time always imply an upper bound on the problem size that may be optimized with this method.

This chapter presents a heuristic approach to the MMGRCPSP as an alternative for when hardware and/or operational time limitations make exact methods computationally unacceptable. The heuristic approach proposed here is an adaptation of an evolutionary algorithm (EA) developed by Sönke Hartmann for the MMRCPPSP [35]. First, Hartmann's EA is presented and extended to MMRCPPSP problems with generalized precedence constraints. Testing is then performed to compare the performance of the EA and the decomposition approach proposed in this dissertation. Finally, a hybrid method, combining the decomposition approach with the EA, is examined in order to decrease solution times and increase the size of problems that can be optimized subject to realistic time and hardware limitations.

### 6.1 *The Evolutionary Algorithm*

A review of the literature yields several evolutionary algorithm implementations for the multi-modal variant of the general resource constrained project scheduling problem [2, 35, 67]. However, evolutionary algorithms have yet to be applied in the literature to problems with both multiple activity execution modes and generalized precedence constraints. Any of the evolutionary algorithm approaches that have been developed for the multi-modal problem could be extended, in part or in whole, to provide an EA solution approach for the MMGRCPSP.

The EA approach developed in this dissertation extends the efforts of Sönke Hartmann [35] since his approach is the most compatible with the solution methodology used in this dissertation. In this section, Hartmann's EA approach is extended to allow for generalized precedence constraints

and to provide  $k$ -best solution sets. It is shown that the mechanics of Hartmann's approach, including the representation of individuals and the operators used on those individuals, can be used directly and only the method for computing the fitness of individuals requires modification.

**6.1.1 Basic Approach.** Hartmann's EA begins by randomly generating an initial population of  $POP$  individuals and then measuring their fitness values. It is assumed that the parameter  $POP$  is an even integer. At this point, the population is randomly partitioned into pairs of individuals and a crossover operator is applied to each pair resulting in two new individuals. Each new individual is subjected to a mutation operator, measured for its fitness value, and added to the population. This increases the population size to  $2 \times POP$ . To obtain the next generation, the selection operator is applied to reduce the population to its original size of  $POP$  individuals, at which point the crossover, mutation, and selection operators are applied again. The population continues to evolve in this manner until the EA reaches either a prespecified CPU time limit or prespecified number of generations,  $GEN$ . Upon termination of the EA, the individual in the final generation with the best fitness value corresponds to the best solution found and the  $k$  individuals with the  $k$ -best fitness values represent the  $k$ -best solutions found. The basic mechanics of Hartmann's EA are illustrated by the flow diagram in Figure 15 [35].

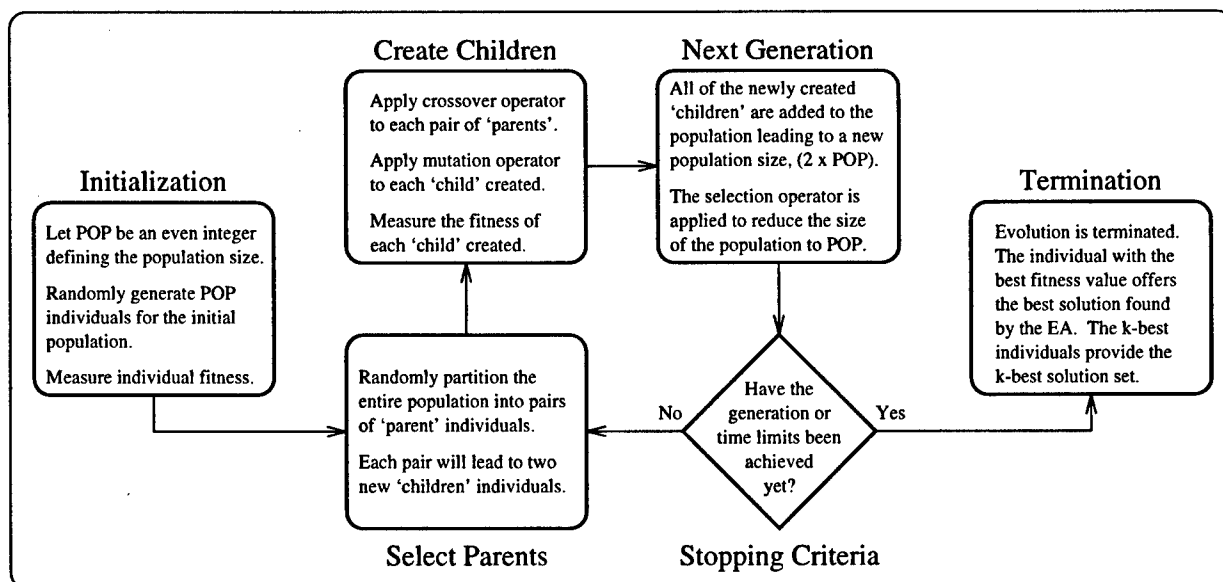


Figure 15 A Flow Diagram of Hartmann's Evolutionary Algorithm

**6.1.2 Individuals.** In Hartmann's EA, individuals are represented by a pair of  $J$ -tuples: one provides a precedence feasible permutation of the set of activities and the other provides a mode assignment for each activity in the set. Recall that these are the same  $J$ -tuples referred to in the definition of an *active permutation schedule* in Chapter IV. Every pair of  $J$ -tuples can be mapped to an *active permutation schedule*. It was proven in Chapter IV that every feasible MMGRCPSP instance must have an optimal *active permutation schedule*. With Hartmann's choice of genetic representation for individuals, his EA stochastically searches the space of all *active permutation schedules*. This is the same space that is deterministically searched by the extended Sprecher algorithm developed in Chapter IV.

To expand on Hartmann's genetic representation of individuals, consider each of the  $J$ -tuples separately. First, consider the activity permutation  $J$ -tuple for some individual  $I$ .

$$\mathcal{G}^I = (g_1^I, g_2^I, \dots, g_J^I)$$

For every individual  $I$ , the activity permutation  $J$ -tuple must be precedence feasible, so if activity  $g_i^I$  is a predecessor of activity  $g_j^I$  then  $i < j$ . Now, consider the mode assignment  $J$ -tuple for individual  $I$ .

$$\mathcal{M}^I = (m_{g_1^I}, m_{g_2^I}, \dots, m_{g_J^I})$$

The value of  $m_{g_i^I}$  is the mode assignment for activity  $g_i^I$  and  $m_{g_i^I} \in M_{g_i^I}$  in order for the mode assignment to be valid. An individual  $I$  is defined by the combination of any precedence feasible activity permutation  $\mathcal{G}^I$  and any valid mode assignment  $\mathcal{M}^I$ . The notation for an individual is given as  $I = (\mathcal{G}^I, \mathcal{M}^I)$ .

The pair of  $J$ -tuples for any individual  $I$  may be mapped to an *active permutation schedule*  $\mathbf{S}^I = (\mathcal{S}^I, \mathcal{M}^I)$  where  $\mathcal{S}^I$  is a set of activity start times  $\mathcal{S}^I = (ST_{g_1^I}, ST_{g_2^I}, \dots, ST_{g_J^I})$ . To derive the set of activity start times, activity  $g_1^I$  is scheduled to begin execution in mode  $m_{g_1^I}$  at time  $ST_{g_1^I} = 0$  and the remaining activities are scheduled in the order given by  $\mathcal{G}^I$ . For  $i = 2, \dots, J$ , activity  $g_i^I$  is scheduled to begin execution in mode  $m_{g_i^I}$  at the earliest possible time subject to precedence and renewable resource constraints. Note that the mode assignment  $\mathcal{M}^I$  is already decided and is not guaranteed to be feasible with respect to the nonrenewable resource constraints so the resulting *active permutation schedule*  $\mathbf{S}^I$  is not necessarily a feasible schedule. Define  $\varphi$

to be an operator that maps an individual  $I$  to an *active permutation schedule*  $\mathbf{S}^I$  in the manner described ( $\varphi(I) = \varphi(\mathcal{G}^I, \mathcal{M}^I) = \mathbf{S}^I$ ).

The fitness value for an individual in Hartmann's EA is a measure of the quality of the schedule provided by  $\varphi(I)$ . The objective for the MMRCPPSP instances that Hartmann worked with was to minimize makespan so in his EA, smaller fitness values correspond to better solutions. For individuals where the schedule provided by  $\varphi(I)$  is feasible, the fitness value for the individual is simply set equal to the makespan of the schedule  $MS(\mathbf{S}^I)$ . The fitness values for individuals where  $\varphi(I)$  produces an infeasible schedule are computed based on the quantity of additional nonrenewable resource that would be required to make the schedule feasible.

Let  $X_k^I$  be the number of additional units of nonrenewable resource  $k$  required for schedule  $\mathbf{S}^I$  to be feasible and let  $\bar{T}$  be the upper bound on the optimal makespan as defined in Chapter IV. Hartmann's complete fitness function is given as follows.

$$f(I) = \begin{cases} MS(\mathbf{S}^I) & \text{if } \mathbf{S}^I \text{ is feasible;} \\ \bar{T} + \sum_{k \in K} X_k^I & \text{if } \mathbf{S}^I \text{ is infeasible.} \end{cases}$$

Note that since  $\bar{T} \geq MS(\mathbf{S}^I) \quad \forall I$ , the fitness value for an individual with a feasible schedule is always smaller than the fitness value for an individual with an infeasible schedule.

It was proven in Chapter IV that, for any regular measure of performance, every feasible MMGRCPSP instance must have an *active permutation schedule* that is optimal. This is the justification for extending Hartmann's EA in the same manner that Sprecher's implicit enumeration algorithm was extended to fit the MMGRCPSP formulation of this dissertation. Aside from the impact of general precedence on the mechanics of the  $\varphi$  operator, the major extension of Hartmann's approach is a modification of the fitness function.

The objective function of the Sweeney-Murphy subproblems, described in Chapter V, is makespan plus weighted resource utilization. According to the definition provided in Chapter IV, this objective function qualifies as a regular measure of performance. For this reason, the Sweeney-Murphy subproblems may be solved with Hartmann's EA after a suitable reformulation of the fitness function. Let  $Y_k^I$  be the total quantity of nonrenewable resource  $k$  used in the schedule  $\mathbf{S}^I$ , let  $\bar{Y}_k^I$  be the maximum possible value of  $Y_k^I$ , and let  $\mu_k$  be the Lagrangian multiplier associated

with the  $k^{th}$  nonrenewable resource (see Chapter V for a discussion of the Lagrangian multipliers in the Sweeney-Murphy subproblems). The reformulation of Hartmann's fitness function is given as follows.

$$f(I) = \begin{cases} MS(S^I) + \sum_{k \in K} \mu_k Y_k^I & \text{if } S^I \text{ is feasible;} \\ \bar{T} + \sum_{k \in K} \mu_k \bar{Y}_k^I + \sum_{k \in K} \mu_k X_k^I & \text{if } S^I \text{ is infeasible.} \end{cases}$$

Since  $\bar{T} + \sum_{k \in K} \mu_k \bar{Y}_k^I \geq MS(S^I) + \sum_{k \in K} \mu_k Y_k^I \quad \forall I$ , the fitness value for an individual with a feasible schedule is always smaller than the fitness value for an individual with an infeasible schedule. The relative fitness of individuals that have feasible schedules is determined by the value of the Sweeney-Murphy subproblem objective function. Among individuals with infeasible schedules, relative fitness is determined by a weighted sum of the additional resources that would be required to make the schedule feasible.

**6.1.3 Operators.** This section presents the three genetic operators used in Hartmann's EA: crossover, mutation, and selection. In the discussion of these operators, an equivalent array representation of individuals is used interchangeably with the  $I = (\mathcal{G}^I, \mathcal{M}^I)$  notation introduced in the previous section. Recall the prior notation.

$$I = (\mathcal{G}^I, \mathcal{M}^I) = ((g_1^I, g_2^I, \dots, g_J^I), (m_{g_1^I}, m_{g_2^I}, \dots, m_{g_J^I}))$$

The new notation displays the same values, but in the form of an array.

$$I = \begin{pmatrix} \mathcal{G}^I \\ \mathcal{M}^I \end{pmatrix} = \begin{pmatrix} g_1^I & g_2^I & \dots & g_J^I \\ m_{g_1^I} & m_{g_2^I} & \dots & m_{g_J^I} \end{pmatrix}$$

The first operator to be discussed is the crossover operator. The details of Hartmann's crossover operator are described in Table 33. The task of this operator is to generate two new individuals by combining the characteristics of two individuals taken from the current population. The individuals selected for crossover are referred to as father  $F$  and mother  $M$ , while the individuals resulting from the crossover are called son  $S$  and daughter  $D$ . The first step of the crossover operator is to generate two random integers,  $p_1$  and  $p_2$ , from the range  $[1, J]$ . The integer  $p_1$  is used

Table 33 Hartmann's Crossover Operator

---

Step 1:	(Randomize) Generate two random integers, $p_1$ and $p_2$ , both in the range $[1, J]$ .
Step 2:	(Derive Child Permutations) For $i = 1, \dots, p_1$ , set $g_i^S = g_i^F$ and $g_i^D = g_i^M$ . For $i = p_1 + 1, \dots, J$ , set $g_i^S = g_j^M$ and $g_i^D = g_k^F$ where $j$ is the lowest index such that $g_j^M \notin \{g_1^S, \dots, g_{i-1}^S\}$ and $k$ is the lowest index such that $g_k^F \notin \{g_1^D, \dots, g_{i-1}^D\}$ .
Step 3:	(Derive Child Mode Assignments) For $i = 1, \dots, p_2$ , set $m_{g_i^S} = m_{g_i^F}$ and $m_{g_i^D} = m_{g_i^M}$ . For $i = p_2 + 1, \dots, J$ , set $m_{g_i^S} = m_{g_i^M}$ and $m_{g_i^D} = m_{g_i^F}$ .

---

in the procedure for generating activity permutations for the children and the integer  $p_2$  is used in the procedure for generating mode assignments for the children.

The second step of the crossover operator is to combine the activity permutations of the parent individuals to form activity permutations for the children. The first  $p_1$  members of the son's activity permutation are taken from the father and the rest are taken from the mother. Likewise, the first  $p_1$  members of the daughter's activity permutation are taken from the mother and the rest are taken from the father. The procedure, described in Step 2 of Table 33, is designed to impart a mix of the characteristics of the father and mother activity permutations while maintaining precedence feasibility [35].

The final step of the crossover operator is to generate mode assignments for each child taken from the mode assignments of the parent individuals. In this step, the random integer  $p_2$  is used in the same way that  $p_1$  was used in generating the activity permutations. The first  $p_2$  mode assignments for the son are taken from the father and the rest are taken from the mother. Likewise, the first  $p_2$  mode assignments for the daughter are taken from the mother and the rest are taken from the father. In this way, each child individual inherits characteristics of both parents.

The following example is taken from Hartmann [35:page 8] to illustrate the application of his crossover operator. The example problem in question includes six activities, each having two possible execution modes. To summarize the precedence conditions, activity 1 is the predecessor of activity 3 which is the predecessor of activity 5 and activity 2 is the predecessor of activity 4 which

is the predecessor of activity 6. Let  $F$  and  $M$  be two individuals from the population of Hartmann's EA when applied to this problem.

$$F = \begin{pmatrix} \mathcal{G}^F \\ \mathcal{M}^F \end{pmatrix} = \begin{pmatrix} 1 & 3 & 2 & 5 & 4 & 6 \\ 1 & 2 & 1 & 1 & 2 & 2 \end{pmatrix} \quad M = \begin{pmatrix} \mathcal{G}^M \\ \mathcal{M}^M \end{pmatrix} = \begin{pmatrix} 2 & 4 & 1 & 6 & 3 & 5 \\ 2 & 2 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Now, if  $p_1 = 3$  and  $p_2 = 4$ , applying the crossover operator to these two parent individuals results in the generation of the following children,  $S$  and  $D$ .

$$S = \begin{pmatrix} \mathcal{G}^S \\ \mathcal{M}^S \end{pmatrix} = \begin{pmatrix} 1 & 3 & 2 & 4 & 6 & 5 \\ 1 & 2 & 1 & 2 & 1 & 1 \end{pmatrix} \quad D = \begin{pmatrix} \mathcal{G}^D \\ \mathcal{M}^D \end{pmatrix} = \begin{pmatrix} 2 & 4 & 1 & 3 & 5 & 6 \\ 2 & 2 & 1 & 1 & 1 & 2 \end{pmatrix}$$

Consider individual  $D$ . Since  $p_1 = 3$ , the first three activities in  $\mathcal{G}^D$  are set the same as the first three activities in  $\mathcal{G}^M$  and the remaining three activities are added to  $\mathcal{G}^D$  in the order that they appear in  $\mathcal{G}^F$ . Now, since  $p_2 = 4$ ,  $\mathcal{M}^D$  is constructed by assigning the first four activities in  $\mathcal{G}^D$  the modes that they were assigned in  $\mathcal{M}^M$  and assigning the last two activities in  $\mathcal{G}^D$  the modes that they were assigned in  $\mathcal{M}^F$ . Note that for both children, the mode assignments are valid and the activity permutations are precedence feasible.

The next operator to be discussed is the mutation operator. The task of this operator is to guide the stochastic search of the EA into regions of the search space that may not be accessible through the crossover of individuals from the current population. Additionally, this operator may help the EA *escape* a local optimal solution. For every individual  $I$ , Hartmann's mutation operator first generates two random integers,  $q_1$  and  $q_2$ , with  $1 \leq q_1 < J$  and  $1 \leq q_2 \leq J$ . The random integer  $q_1$  is used to alter  $\mathcal{G}^I$  and  $q_2$  is used to alter  $\mathcal{M}^I$ .

If the resulting activity permutation is precedence feasible,  $\mathcal{G}^I$  is altered by interchanging activities  $g_{q_1}^I$  and  $g_{q_1+1}^I$ . If the interchange of these activities would result in an activity permutation that is not precedence feasible, the mutation operator does not alter  $\mathcal{G}^I$ . The mutation operator alters the mode assignments in  $\mathcal{M}^I$  by changing the value of  $m_{g_{q_2}^I}$  to assign a new execution mode for activity  $g_{q_2}^I$ . This new mode is selected at random from the set of all possible execution modes for activity  $g_{q_2}^I$ .



The final operator discussed is Hartmann's selection operator. Each generation, Hartmann's EA partitions the entire population into pairs, applies the crossover operator to create two new individuals from each pair, applies the mutation operator to each new individual, and then adds all of the new individuals to the current population. This increases the population size from  $POP$  to  $2 \times POP$ . The next generation is achieved by applying a selection operator to reduce the population back to its original size of  $POP$  individuals.

Hartmann experimented with two different selection operators. The first selection operator was based on simple survival-of-the-fittest. This operator selected the  $POP$  individuals with the best fitness function values as the population of the next generation and discarded the rest. The other selection operator was a stochastic variant of the survival-of-the-fittest approach that gives every individual a chance of making the next generation based on its fitness value. In Hartmann's test results, the deterministic variant significantly outperformed the stochastic variant so the extended Hartmann EA implementation for this dissertation research uses the deterministic selection operator.

## 6.2 Testing the Evolutionary Algorithm

This section presents testing results for the extended Hartmann evolutionary algorithm. This testing is divided into two distinct phases, one phase performs tests to determine the most effective settings for the EA parameters and the other applies the EA approach and the decomposition approach from Chapter V to a common set of test problems in order to compare the resulting solution times. The testing environment is the same as in other parts of this dissertation. The algorithms are coded in C, the standard C function *gettimeofday* is used to obtain solution times, and the test runs are performed on a 486DX66 processor.

**6.2.1 Configuring the Evolutionary Algorithm.** The testing presented in this section is intended to provide empirical results that offer insight into the problem of selecting the most effective settings for the adjustable parameters of the extended Hartman EA. Note that the *optimal* parameter settings obtained through this testing are *optimal* only for the range of the test problems used. The three adjustable parameters that are considered are population size, generation limit, and stopping criteria. The most effective settings for these parameters depends on the type of problem instances to which the algorithm is applied. Since this EA is intended as an alternative for the decomposition

approach discussed in Chapter V, an appropriate fine-tuning test set would be the one used for the testing of the decomposition approach.

The test set used in this section is the one used to obtain the final set of testing results for the decomposition approach in Chapter V. The set includes problem instances with three different levels of total activities and three different levels of granularity. In this case, granularity refers to the number of subproblems in each problem instance. To examine the effect of population size and generation limit on the performance of the EA, each of the 90 problem instances in the test set is solved first with the decomposition approach from Chapter V, and then again with 25 different combinations of population size and generation limit. Table 34 reports the performance of the EA for each combination of population size and generation limit in terms of average deviation from optimal, maximum deviation from optimal, and the percent of problem instances for which the EA obtained the optimal solution.

Table 34 Configuring Population Size and Generation Limit

Total Individuals		2400	4800	7200	9600	12000
<i>POP</i> = 40	Total Generations	60	120	180	240	300
	Average Deviation	2.53%	2.44%	2.37%	2.37%	2.33%
	Maximum Deviation	14.29%	14.00%	12.12%	12.12%	12.12%
	Percent Optimal	51.11%	51.11%	51.11%	51.11%	52.22%
<i>POP</i> = 80	Total Generations	30	60	90	120	150
	Average Deviation	2.16%	1.87%	1.84%	1.84%	1.82%
	Maximum Deviation	12.12%	12.12%	12.12%	12.12%	12.12%
	Percent Optimal	48.89%	52.22%	53.33%	53.33%	53.33%
<i>POP</i> = 120	Total Generations	20	40	60	80	100
	Average Deviation	3.01%	1.72%	1.64%	1.62%	1.62%
	Maximum Deviation	14.00%	9.09%	9.09%	9.09%	9.09%
	Percent Optimal	36.67%	54.44%	57.78%	57.78%	57.78%
<i>POP</i> = 160	Total Generations	15	30	45	60	75
	Average Deviation	4.25%	1.55%	1.49%	1.49%	1.49%
	Maximum Deviation	16.67%	8.57%	8.57%	8.57%	8.57%
	Percent Optimal	26.67%	58.89%	63.33%	63.33%	63.33%
<i>POP</i> = 200	Total Generations	12	24	36	48	60
	Average Deviation	5.85%	2.03%	1.60%	1.46%	1.43%
	Maximum Deviation	22.22%	12.12%	12.12%	9.09%	8.57%
	Percent Optimal	20.00%	53.33%	62.22%	63.33%	63.33%

The combinations of population size and generation limit tested and reported in Table 34 were selected in the following manner. First, a range of potential population sizes were selected. The

range of population sizes selected covers the range used by Hartmann, but it includes some larger values since Hartmann's test problems have only 10 to 20 activities while the problems for testing the extended Hartmann EA have between 24 and 48 activities [35:page 13]. Next, a range of values for the total number of individuals to generate was selected. These values are not required by the EA, but are used to obtain the generation limits for each test case. Hartmann used 3000, but did not provide any justification for this value [35:page 13]. Assuming 3000 was a valid choice for the size of the test problems Hartmann used, this testing used a range of values, 2400 to 12000, that covers the value used by Hartmann, but extends to higher values to account for the larger size of the test problems used here. Finally, a generation limit for each combination of population size and total number of individuals was computed such that  $GEN = INDIV/POP$ .

In Table 34, each entry in a given column reports the results for the EA with different settings for population size and generation limit, but the same total number of individuals is generated in each case ( $GEN \times POP = INDIV$ ). The idea is that since each of the configurations reported in a given column generates the same total number of individuals, the solution times should be approximately the same, but from column to column there is a definite difference in solution times. To select the most effective parameter settings, the best configuration from each column is identified according to the lowest average deviation from optimal. The graph in Figure 16 plots the average deviation from optimal versus the total number of individuals for the best configuration from each column.

The testing results in Table 34 and Figure 16 make a strong argument to use a population size of 160 and a generation limit of 45 when applying the EA to problem instances with characteristics similar to the test problems. First, consider the graph in Figure 16. The slope of the line in this graph is a measure of the improvement in the performance of the EA attained through increases in the total number of individuals generated. The greater the slope, the greater the improvement in performance. As the line levels off, further increases in the total number of individuals, and the corresponding increases in solution times, achieve only minor increases in performance. The goal is to choose the value for the total number of individuals at the point where the slope levels off. The two points where there is reduction in slope on the graph are at 4800 and 7200 total individuals.

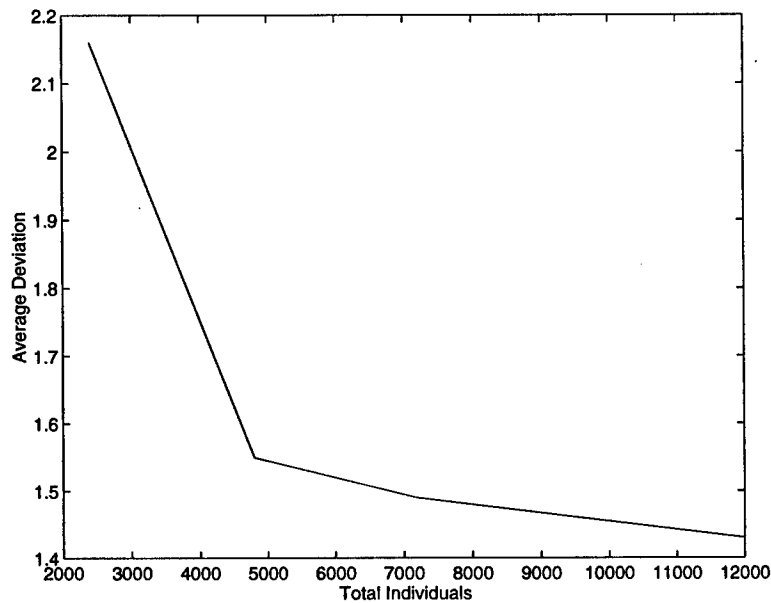


Figure 16 Average Deviation versus Total Individuals

Now, consider the columns in Table 34 that correspond to these points where there is a reduction in slope. For the columns associated with 4800 and 7200 total individuals, in both cases the best configuration has a population size of 160. The configuration for 4800 total individuals has a generation limit of 30 and the configuration for 7200 total individuals has a generation limit of 45. Based on the performance data for these two configurations, the *standard* EA configuration is set to a population size of 160 and a generation limit of 45. Not only is this configuration non-dominated among the set of all configurations with a population size of 160, but its maximum deviation and percent optimal performance values are non-dominated in the entire set of configurations included in the test.

The last adjustable parameter to be set is the stopping criteria for the EA. So far, the only stopping criteria used was a generation limit. The population continues to evolve until a predetermined number of generations has been reached, at which point, the EA terminates. Now that the population size and generation limit have been set to a *standard* configuration, a secondary stopping criteria is introduced. The quality of a population is measured by the fitness value of its best individual. The new stopping criteria terminates evolution if the quality of the population remains unchanged for a certain number of generations. The next set of tests is used to determine

what threshold value is most effective for the new stopping criteria. That is, how many generations without improvement are allowed before the EA is terminated.

The next set of tests examines the performance of the EA with stopping criteria threshold values of 3, 6, 9, 12, and 15 generations without improvement. The EA is applied to all 90 of the test problems used previously, once for each of the possible threshold values. To evaluate the effect of the threshold value on the performance of the EA, an average deviation from optimal value is computed for each threshold value from the test results. The test results are displayed in Figure 17 as a graph with the threshold values plotted against the corresponding average deviation from optimal values.

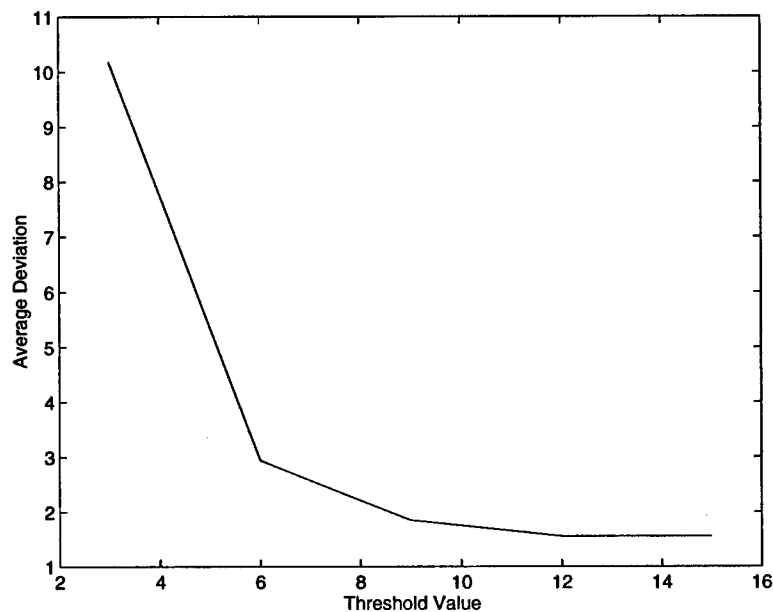


Figure 17 Stopping Criteria Threshold Test Results

There is an implicit tradeoff in the graph in Figure 17. Higher threshold values allow more unimproved generations before termination. The extra generations offer the potential for further improvement at the cost of longer solution times. Again, the goal is to select the point where the slope of the line in the graph levels off and the probability of further improvement is low. Based on the test results shown in the graph, the *standard* stopping criteria threshold value is set to nine. That is, the EA terminates after nine generations with no improvement in the quality of the population.

Unless otherwise noted, all further applications of the extended Hartmann EA use a configuration with a population size of 160, a generation limit of 45, and a stopping criteria threshold value of nine. It is important to note that the testing approach used in this section selected these EA parameters in order to optimize performance on a set of test problems. There are no performance guarantees for problems not included in the test set. Before applying the EA to problems with characteristics significantly different from the problems in the test set, it would be beneficial to re-tune the EA parameters.

*6.2.2 Comparing the Evolutionary Algorithm to Sweeney-Murphy.* The testing presented in this section compares the solution times of the EA with the solution times of the decomposition method from Chapter V on problems of varying sizes. The purpose of this testing is twofold. First, the tests show the value of a good heuristic approach for larger problems since worst case solution time is exponentially related to problem size for the exact algorithms. Second, the tests provide empirical data to illustrate the exponential nature of the decomposition approach and to explore how big is too big given the limitations of time and hardware.

The set of problems for this testing is similar to the problems used in the previous section with one exception; there are exactly three subproblems in every problem instance. Only the total number of activities is varied. The levels of total number of activities included in the test set are 30, 45, 60, 75, and 90, and 10 problem instances are generated for each level. This generates a test set of 50 problem instances ranging from 30 to 90 total activities. Both the EA and decomposition approaches are applied to all 50 problem instances and a time limit of two hours is imposed for the decomposition approach to solve any one problem instance. The results are given as average solution time in seconds and are summarized in Table 35.

Table 35 Comparing the Performance of Decomposition and EA

Total Activities	30	45	60	75	90
Evolutionary Algorithm	5.50	6.20	8.68	11.18	14.24
Decomposition Approach	0.98	23.89	200.08	3206.22	N/A
Average Deviation from Optimal	1.25%	2.04%	1.46%	1.59%	N/A

For each level of total activities, the data in the last row of Table 35 shows the average deviation of the EA solutions from the optimal solutions obtained by the Sweeney-Murphy algorithm. The

average deviation data suggests that, at least for the range of problem sizes that can be solved by the decomposition algorithm, the EA can provide solutions that are, on the average, within approximately two percent of optimal. This observation applies to problem instances that conform to the characteristics of the problems in the test set that was used for fine tuning the EA.

The exponential relationship between problem size and solution time for the decomposition approach is evident in the results summarized in Table 35. For the smaller problem instances, the decomposition approach actually outperforms the EA, but the average solution time for the decomposition approach increases by an order of magnitude each time the problem size is increased by 15 total activities. One of the problem instances with 75 total activities could not be solved within the imposed two hour time limit, and none of the 90 activity problems could be solved in the time allowed.

These results indicate a limitation on the size of a problem that may be solved by the decomposition approach given hardware and time limitations. Note, however, that the apparent 75 activity upper bound on problem size is for problem instances with three subproblems. The actual limitation is in the subproblem solution algorithm for which solution time explodes shortly after 25 activities and hence, the 75 total activity limit for problems with three subproblems. Problems with more subproblems have a correspondingly larger limit on the total number of activities.

### 6.3 *A Hybrid Approach*

This section presents a hybrid solution approach for the MMGRCPSP that combines the extended Hartmann EA approach with the decomposition approach from Chapter V. There are several possibilities for such a combination, but the approach presented here is one that generates optimal solutions. The computational effort of the decomposition approach of Chapter V can be partitioned into three distinct categories: subproblem solution, master problem solution, and the computation of Lagrangian multipliers. Of these three categories, the third, computing the Lagrangian multipliers, is the only task that may be accomplished by an EA without resulting in a heuristic solution approach, sacrificing the guarantee of optimality.

Once the Sweeney-Murphy decomposition optimality criteria have been met, the optimal solution to the master problem is also optimal for the original problem. If an EA approach was used

to solve the master problem, there would be no guarantee that the optimal master problem solution has been found and the resulting approach could only be used as a heuristic. The optimality criteria for the Sweeney-Murphy algorithm are only valid if the  $k$ -best solutions for every subproblem are generated. An EA could be used to generate  $k$  *good* solutions, but there is no guarantee that those  $k$  solutions are the best. If an EA was used for the subproblems in a Sweeney-Murphy decomposition approach, the lack of  $k$ -best solution sets would invalidate the optimality criteria and the resulting solution approach could only be used as a heuristic. The Sweeney-Murphy approach is guaranteed to converge for any value of the Lagrangian multipliers. The quality of the multiplier values used affects only the solution time of the algorithm. For this reason, an EA could be used to obtain Lagrangian multipliers without sacrificing the guarantee of optimality.

The hybrid approach proposed in this section implements a method that uses the extended Hartmann EA to obtain values for the Lagrangian multipliers required by the Sweeney-Murphy subproblems. Recall the three problem formulations introduced in Chapter V: the main block-angular integer program  $\mathbf{P}$ , the Sweeney-Murphy subproblems  $\mathbf{SP}_i(\mu)$ , and the Sweeney-Murphy master problem  $\mathbf{MP}$ . These formulations are referred to in the discussion of the hybrid solution approach. In addition to these formulations, the parameters  $K$ , the number of problem resources,  $P$ , the number of subproblems, and  $A$ , the number of activities in each subproblem, are also important to the discussion.

The decomposition approach developed in Chapter V built upon a parametric analysis approach originally proposed by Nauss [64] to compute *proxy dual variables* to be used as Lagrangian multipliers. Nauss' approach calls for an integer program to be solved  $K + 1$  times in order to estimate the *marginal benefit* of  $K$  problem resources. Chapter V presents the reasons why this approach could not be used and offers an alternate method of parametric analysis that can be used to calculate the necessary *proxy dual variables*. The disadvantage of the alternate approach is that the computational effort involved grows exponentially with problem size and serves to further limit the size of problems that may be solved under realistic time constraints.

The first step of the alternate approach for finding *proxy dual variables* calls for a feasible allocation of problem resources between the  $P$  subproblems to be found. Once such an allocation is made, the linking constraints may be dropped and the resulting subproblems are truly independent.



The solutions to these subproblems may be combined and the result is a feasible solution to the original problem **P**.

To obtain the marginal benefit of a resource, the parametric analysis described in Chapter V requires that each of the  $P$  subproblems be solved once with the original allocation of resources, and then again with the constraint for the resource in question relaxed. The constrained subproblem solutions combine to form a feasible solution to **P** and the unconstrained subproblem solutions form a solution to a relaxed formulation of **P**. A comparison of these two solutions to **P** yields a value for the amount of additional resources used and for the amount of objective function improvement provided by the additional resources. If both of these values are non-zero, the *proxy dual variable* is given as the ratio between the objective function improvement and the additional resources required.

This approach requires  $P \times (K + 1)$  problems of  $A$  activities each to be solved in order to estimate the marginal benefit of the  $K$  problem resources. These problems are solved using the extended Sprecher algorithm from Chapter IV. Since the average solution time for the extended Sprecher algorithm is exponentially related to the number of activities in the problem, the computational effort involved in finding the *proxy dual variables* becomes significant as  $A$  gets large.

The computational effort involved in calculating marginal benefit values as *proxy dual variables* for large problem instances may be significantly reduced by using Nauss' parametric analysis approach with the EA developed in this chapter. The new approach consists of solving the original problem **P**  $K + 1$  times with the EA, once with all of the constraints in force, and then  $K$  times more, each time relaxing one of the  $K$  resource constraints. Once again, the marginal benefit of each resource is estimated as the ratio of the objective function improvement to the additional resources required.

The mechanical difference between the original and the new EA approach for calculating marginal benefit is in the size, number, and type of solutions that are required. The original approach requires optimal solutions for  $P \times (K + 1)$  problems of  $A$  activities each while the new EA approach requires heuristic solutions for  $K + 1$  problems of  $P \times A$  activities each. Aside from the mechanical difference, there may also be a difference in the quality of the Lagrangian multipliers obtained by the two approaches. The quality of the multipliers obtained by the original approach may vary based on how the problem resources are allocated among the subproblems. Similarly, the

quality of the multipliers obtained with the new EA approach may vary depending on how close the heuristic solutions come to optimality for a given problem instance.

The mechanical and quality differences between the two approaches affect how the solution times of the hybrid decomposition approach compare to the standard decomposition approach for a given problem size. It is difficult to predict how some of these differences affect the solution time of the decomposition, but one thing is certain; as problem size increases, the exponential effort required by the original approach eventually reaches a point where the hybrid approach offers a significant reduction in solution time.

In the following test, the standard and hybrid variants of the decomposition approach from Chapter V are applied to a common set of test problems. The test set used is the same set of test problems used earlier to compare the performance of the EA to the performance of the decomposition approach. Since both the standard and hybrid decomposition approaches obtain optimal solutions, the results given in Table 36 report only the average solution time in seconds.

Table 36 Comparing Hybrid and Standard Decomposition Approaches

Total Activities	30	45	60	75
Hybrid Decomposition	9.16	32.66	139.41	1321.28
Standard Decomposition	0.98	23.89	200.08	3206.22

The results in Table 36 show that, as expected, the hybrid approach offers significant reduction of solution times for larger problem instances. The standard approach does better on smaller problems due to the small size of the subproblems, but as problem size grows it is computationally tasking to solve  $P \times (K + 1)$  larger subproblems before the Sweeney-Murphy decomposition can even begin. It should be noted that some of the solution time improvement may also be due to an improved quality of Lagrangian multipliers. Better multipliers lead to fewer Sweeney-Murphy iterations which means faster solution times.

Figure 18 plots solution time against the total number of activities for both the standard and hybrid decomposition approaches. Since there is an exponential relationship, the solution times are plotted in logarithmic scale. The graph in Figure 18 illustrates that the merits of the hybrid approach increase with problem size. The apparent breakpoint is somewhere between 45 and 60 total activities, but it is important to note that this breakpoint is for problems with the

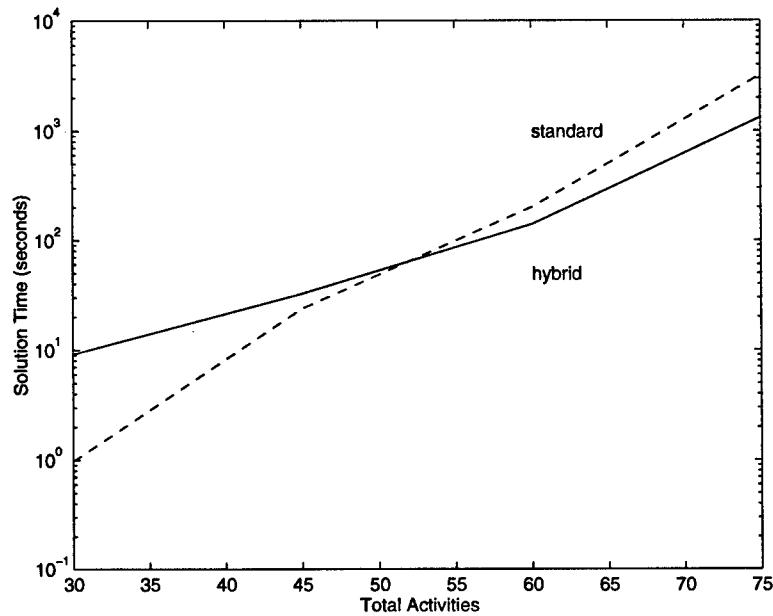


Figure 18 Standard Versus Hybrid Decomposition

same characteristics as those in the test set. Specifically, all of the test problems had exactly three subproblems ( $P = 3$ ). The breakpoint varies for problems with other values of  $P$ , but the exponential nature of the original approach guarantees that a break point must exist.

#### 6.4 Summary

This chapter developed an evolutionary algorithm for the MMGRCPSP. This heuristic approach serves as an alternative for when time or hardware limitations prohibit the use of an exact solution procedure. The evolutionary algorithm was used in conjunction with the decomposition algorithm in Chapter V to develop a hybrid decomposition approach that reduces solution time for some larger problem instances. This hybrid approach is applied in a case study in Chapter VII to demonstrate the applicability of MMGRCPSP models and solution methodologies to the force level air campaign planning problem.

## VII. Case Study

The purpose of this chapter is to apply the specialized large-scale integer program decomposition algorithm developed in Chapters IV, V, and VI to a sample combat planning problem in order to demonstrate the applicability of the approach. This demonstration includes examples of how decomposition may be applied to a combat planning problem, how generalized precedence constraints may be used to enforce combat mission timing, and how doubly constrained resource constraints may be used to restrict aircraft and sortie utilization. The combat planning model developed in this dissertation effort was designed to be flexible to allow for the varied decomposition, mission timing, and resource accounting requirements of different operational scenarios. This demonstration is intended as a proof of concept only. The applicability of the approach is not limited to the examples presented here.

This chapter consists of two sections. The first section describes how the case study sample problem was generated. The complete set of data required to formulate the sample problem is reported in Appendix B. The second section reports the results of applying the solution methodology developed in this dissertation to the case study sample problem. A  $k$ -best solution set for the sample problem is presented and several criteria are offered for choosing between several near optimal solutions.

### 7.1 Problem Generation

The three components necessary to formulate the model proposed in this dissertation for any combat planning scenario are activities, resources, and precedence relations. The activity and resource information may be taken directly from basic scenario data. The asset and target information of a scenario dictates the activities and resources in the formulation of that scenario. However, the precedence relations specified for the formulation of a combat planning scenario are driven by military strategy and may not be taken directly from scenario data. These relations must be provided by an operational expert.

**7.1.1 Scenario.** The first issue to address in the generation of a sample problem is the operational scenario underlying the combat planning problem being formulated. The generation of the sample problem begins with the map illustrated in Figure 19. The map in Figure 19 provides the

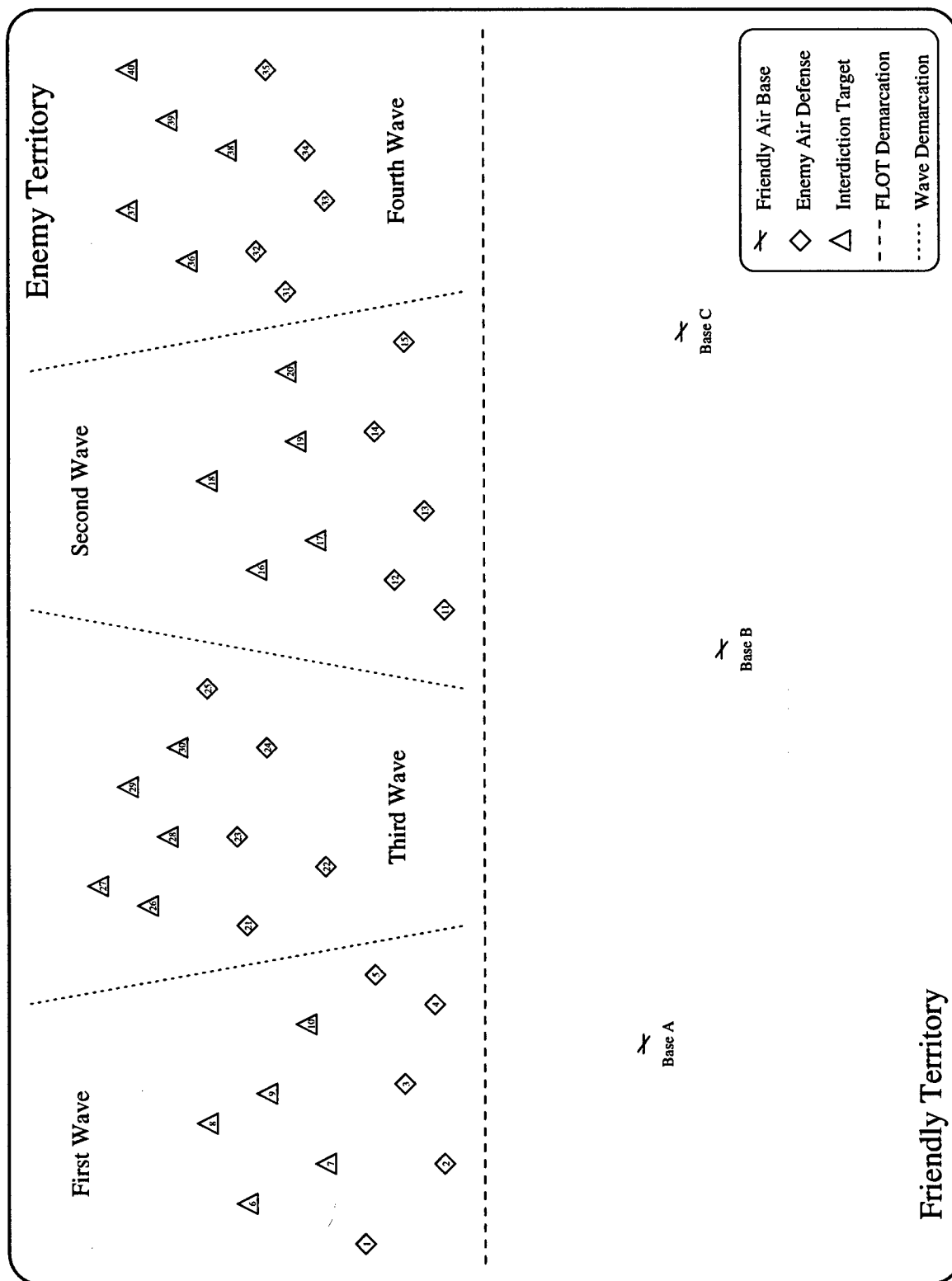


Figure 19 A Map for the Case Study Scenario

location of all of the assets and targets that are included in the formulation of the sample problem. The general approach taken in the development of the illustrative sample problem is to stay as simple and generic as possible, and this general philosophy is apparent in the map.

The map is simple in that it only contains features absolutely necessary to formulate a problem. One such necessary feature is the dashed line representing the forward line of own troops or FLOT. This is a line of demarcation between friendly and enemy forces and is used to mark the point at which a mission enters enemy airspace and is subject to the threat of enemy air defenses. The FLOT for a scenario could take on any shape depending on the actual positioning of forces. A straight line has been used in this example to make routing calculations easier.

The map is generic in that there are no specific names given for any of its features. Bases are referred to as Base X, the only label on the targets is an identification number, and territories have no titles beyond friendly and enemy. The one further distinction among targets beyond the simple identification number is the target type. The two target categories used are enemy air defense and interdiction. The enemy air defense targets represent the possible threats to friendly aircraft, like surface to air missile sites. The interdiction targets are targets identified to reduced the enemy's ability to continue the war; items such as headquarters, bases, or supply depots. The generic nature of the sample problem is intended to keep the focus concentrated on the application of the solution methodology and the usefulness of the results. Political and military implications of the scenario are therefore excluded.

In addition to the location of targets and assets, the map in Figure 19 illustrates one method of how the problem may be decomposed into semi-independent subproblems. To decompose this or any other problem, the underlying problem structure must be examined for evidence of block-angular structure. The idea is to partition the activities into discrete sets such that the number of linking constraints is kept relatively small. In the case of combat planning, the discriminating activity feature could be the type/location of resources required, the type/location of the target associated with the activity, or the timing of the resulting missions. For the sample problem generated here, the activities are partitioned based on two different criteria.

First, the activities are partitioned according to physical location. The dotted lines on the map in Figure 19 partition the target locations into four discrete groups. These groups of targets

are attacked in temporally discrete waves, so they are partitioned by time as well as location. On the map, the waves are labeled according to their timing which was decided according to the proximity of the targets that make up the wave to the FLOT. The objective of the problem is to minimize the exposure to enemy air defenses for each wave. For a given wave, this value is calculated as the difference between the time that the first mission in the wave crosses the FLOT to enter enemy territory and the time that the last mission in the wave crosses the FLOT leaving enemy territory. The problem's objective function is the sum of the exposure values for the individual waves.

The second form of activity partitioning that is used in the generation of this sample problem is based on target type. Half of the target locations in each wave are enemy air defenses and the other half are interdiction type targets. When generating the precedence constraints that control the relative timing of the missions in each wave, an assumption is made that all enemy air defense targets must be taken out before the deeper, interdiction type missions cross into enemy air space. This creates a bottleneck in the underlying precedence network for each wave with the enemy air defense targets on one side and the interdiction targets on the other. This bottleneck offers a good breakpoint to decompose the problem further by separating the air defense and interdiction targets in each wave into separate subproblems. With four waves and two subproblems for each wave, the formulation of the sample problem has a total of eight subproblems.

*7.1.2 Activities and Resources.* The map in Figure 19 shows a total of 40 target locations, half of them enemy air defense targets and the other half interdiction targets. In the sample problem, it is assumed that each target location includes two or three separate targets that must be attacked by separate missions. For instance, if the target is an enemy air base it may be necessary to designate one mission to crater the runway, one to take out the control tower, and another to destroy aircraft on the ground. It is assumed that each enemy air defense target location has two separate targets to hit and each interdiction target location has three. With 20 enemy air defense targets locations having two separate targets and 20 interdiction targets having three separate targets, the sample problem has a total of 100 targets and the problem formulation has a corresponding total of 100 activities. An attack mission of either two or four aircraft must be planned against each of the 100 targets, resulting in an ATO of 200-400 sorties.

To formulate the problem, it is necessary to determine modes, durations, and resource requirements for each of the 100 activities in the problem. In an actual combat planning scenario, the information necessary to determine the modes and resource requirements would be found in the target nomination list (TNL). To specify modes and resources for the sample problem, a table of data similar to a TNL is generated. This table includes only that subset of the actual TNL information that is essential to determining modes and resource requirements for the sample problem activities.

For this sample problem it is assumed that there are two types of aircraft available to fly attack missions, AC-1 and AC-2. Additionally, it is assumed that there are two units of each aircraft type available for the scenario. One AC-1 unit is located at Base A, the other at Base B. One AC-2 unit is located at Base B and one is located at Base C. Modes and resource requirements for the sample problem activities are assigned randomly. Some targets can be attacked with either aircraft type while other targets require one or the other. This means that every activity has either two or four possible execution modes.

The TNL for the sample problem specifies the number, type, and location of resources required for every mode of every target at every target location. The number of sorties called for by a given mode is randomly determined to be either two or four. An example of the resources required for a given mode of some target might be two sorties of AC-1 from Base B. The entire TNL for the sample problem is provided in Appendix B and some sample entries are shown in Table 37.

Table 37 Sample TNL Entries

LOC	X	Y	Target Type	NTAR	TAR	NMOD	MOD	Num/Type Resources
5	31	59	Air Defense	2	9	2	1	4 AC-1 from Base A
							2	4 AC-1 from Base B
					10	4	1	4 AC-1 from Base A
							2	4 AC-1 from Base B
6	8	72	Interdiction	3	11	2	3	4 AC-2 from Base B
							4	4 AC-2 from Base C
							1	2 AC-2 from Base B
							2	2 AC-2 from Base C
					12	4	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
							3	2 AC-2 from Base B
							4	2 AC-2 from Base C
					13	2	1	2 AC-1 from Base A
							2	2 AC-1 from Base B



The first five columns in Table 37 provide information about a target's location, specifically the identification number from the map, the (x,y) coordinates of the location, and the type and number of targets located at those coordinates. The next two columns offer information about each of the targets at a given location including a target identification number and the number of possible execution modes for that target. The last two columns give the details for every execution mode which consist of a mode identification number and the number, type, and location of the necessary resources. For example, target 11 is an interdiction target at location 6 and must be attacked with two AC-2 from either Base B or Base C.

The complete TNL provides all the information necessary to determine the modes and resource requirements of all activities in the sample problem. The final information necessary to completely specify all of the activities are the durations associated with each execution mode. Before this information can be computed, more resource information must be specified. Table 38 provides the additional details necessary for computing mode durations.

Table 38 Problem Resource Information

Unit	Type/Base	X	Y	NAC	TR	NSOR	TT	NAS
1	AC-1 from Base A	24	32	36	3.0	108	50	1.0
2	AC-1 from Base B	64	24	24	3.0	72	50	1.0
3	AC-2 from Base B	64	24	24	2.5	60	70	1.1
4	AC-2 from Base C	96	28	36	2.5	90	70	1.1

Table 38 provides the (x,y) coordinates of the base at which each unit is located. In addition, the table provides some information about the aircraft that make up the units. The NAC column gives the number of aircraft in the unit, the TR column gives the turn rate, or number of sorties per day per aircraft, and the NSOR column gives the total number of sorties each unit can perform in a given ATO day ( $NAC \times TR = NSOR$ ). The TT column gives the aircraft turn time which specifies the amount of time it takes to service aircraft after they have completed a mission. This value is given in minutes.

The final column in Table 38 contains a nominal airspeed value for each aircraft type and is given in distance units per minute. This value is set to make the average in flight time for missions in the sample problem equal to approximately two hours. The (x,y) coordinates of the bases and targets are based on a grid where each distance unit is an eighth of an inch on the map. The

average round trip distance between the bases and targets specified in the TNL is approximately 120 distance units. A nominal airspeed of 1.0 distance units per minute makes the average in flight time of a mission two hours, and with an average turn time of one hour, the average mission ties up assets for approximately three hours or 180 minutes. To make the two aircraft types different, AC-2 is given a faster nominal airspeed in exchange for a smaller turn rate and a longer turn time.

With the additional resource information in Table 38, it is now possible to compute mode durations for the activities in the sample problem. Activity execution mode durations represent the duration of a combat mission, including aircraft turn time, for a given asset/target combination. In order to formulate generalized precedence constraints to enforce mission timing, it is necessary to know the duration of each leg of a mission as well as the entire mission duration (the sum of all mission leg durations).

Every mission in the sample problem has six component mission legs. The leg from the base to the FLOT is referred to here as the approach leg (APPR) and the leg from the FLOT to the target area is called the ingress leg (INGR). The third leg is called the attack leg (ATT) and is a short period of time that the target area belongs to one mission. The duration of this leg is fixed to one minute for all missions. The purpose of this leg is to avoid interference between separate missions to the same target location. Note that the duration of the attack leg is not intended as loiter time. The leg from the target area back to the FLOT is called the egress leg (EGR) and the leg from the FLOT back to the base is called the return leg (RET).

The final mission leg is the turn time (TURN) for the aircraft to be serviced after the mission is complete. Although all of the missions in the sample problem have six legs, missions in other scenarios could include more than just the basic components described here. For instance, if a mission requires in flight refueling to reach the target, the approach leg would be replaced by a leg from the base to a tanker airspace, a loiter time to refuel, and a leg from the tanker airspace to the FLOT. Additional mission legs require more mode duration computations, but do not increase the solution time of the algorithm.

Two of the six mission leg durations are already known. The turn times are provided in Table 38 and the attack leg is fixed at one minute for all missions. To determine the durations of the remaining four mission legs for each activity execution mode, simple geometry is used to compute

the distance of each leg and the distance is divided by the nominal airspeed of the aircraft called for by the execution mode. Straight line routing is used to determine the necessary distances. Note that this is an acceptable method for calculating durations, but the actual missions flown use neither straight line routes nor nominal airspeeds. Actual routing is accomplished according to terrain and threats, and airspeed is regulated in order to reach mission waypoints at the appropriate times.

Once durations are computed for all of the components of every activity execution mode, the complete mission duration is given as the sum of all component durations. All of the duration information for every activity in the sample problem is given in Appendix B and some samples are shown in Table 39.

Table 39 Sample Duration Information

LOC	TAR	MOD	APP	ING	ATT	EGR	RET	TUR	SUM
5	9	1	17	11	1	11	17	50	107
		2	33	15	1	15	33	50	147
	10	1	17	11	1	11	17	50	107
		2	33	15	1	15	33	50	147
		3	30	14	1	14	30	70	159
6		4	42	23	1	23	42	70	201
		1	33	33	1	33	33	70	203
		2	40	48	1	48	40	70	247
		1	17	26	1	26	17	50	137
	12	2	37	37	1	37	37	50	199
		3	33	33	1	33	33	70	203
		4	40	48	1	48	40	70	247
	13	1	17	26	1	26	17	50	137
		2	37	37	1	37	37	50	199

The duration information in Table 39 is for the same set of targets whose TNL entries were displayed in Table 37. All duration values are reported in minutes. Note that the last column in Table 39 is the complete mission duration for a given activity execution mode. By design, the average of the values reported in this column should be approximately 180 minutes or three hours. All of the activity information necessary to formulate the sample problem is found in either the TNL or in the activity duration table and all of the required resource information is reported in Table 38.

**7.1.3 Precedence.** The final component necessary to formulate the sample problem is a set of activity precedence constraints. Unlike the activity and resource information, the precedence

information cannot be simply extracted from standard scenario information. How to sequence missions in order to gain the best results is a matter of operational art and a task best left to combat planning experts. For the sample problem, several mission sequencing alternatives are presented and applied. These alternatives may not be the same mission sequencing that an operational expert would use, but they are intended to demonstrate how generalized precedence is used. Once the mechanics are understood, the constraints can be used to implement any type of mission sequencing that is required.

Figure 20 illustrates the six types of activity precedence relations that are used in the sample problem formulation. The basic formulation of the generalized precedence constraints used in this dissertation may be summarized as  $ST_j - ST_i \geq \Delta_{ijmn}$  where  $ST_i$  is the start time of the predecessor,  $ST_j$  is the start time of the successor, and  $\Delta_{ijmn}$  is a minimum lag value that is dependent on the modes of both  $i$  and  $j$ . All six of the precedence relations depicted in Figure 20 are formulated in the same manner with the lone difference being the method for calculating the minimum lag value.

In the illustrations in Figure 20,  $i$  indicates a predecessor activity,  $j$  indicates a successor activity,  $s$  indicates a dummy source activity,  $t$  indicates a dummy sink activity, and  $ST(x)$  indicates the start time of activity  $x$ . The diagram for each type of precedence includes a predecessor activity, either  $i$  or  $s$ , and a successor activity, either  $j$  or  $t$ . An arrow emanates from each successor activity at a critical point in the duration of the activity and points to the right to indicate that that point in the activity must occur at that time (relative to the predecessor) or later. In addition, the minimum lag value necessary to enforce a precedence condition is illustrated in each precedence diagram.

Each enemy air defense subproblem in the sample problem includes a zero duration dummy source activity. This activity becomes the predecessor of every activity in the subproblem that does not already have a predecessor. Type A precedence is used to define the precedence relation between this dummy activity and each of its successors. The minimum lag time used for this type of precedence is zero. This allows the successors of the dummy activity to begin execution at the same time as the dummy.

Type B precedence is used to sequence missions against targets at the same location. If there are two targets at a location, they must be taken out sequentially in order to avoid interference between the missions. The second mission cannot begin its attack leg until the attack leg of the

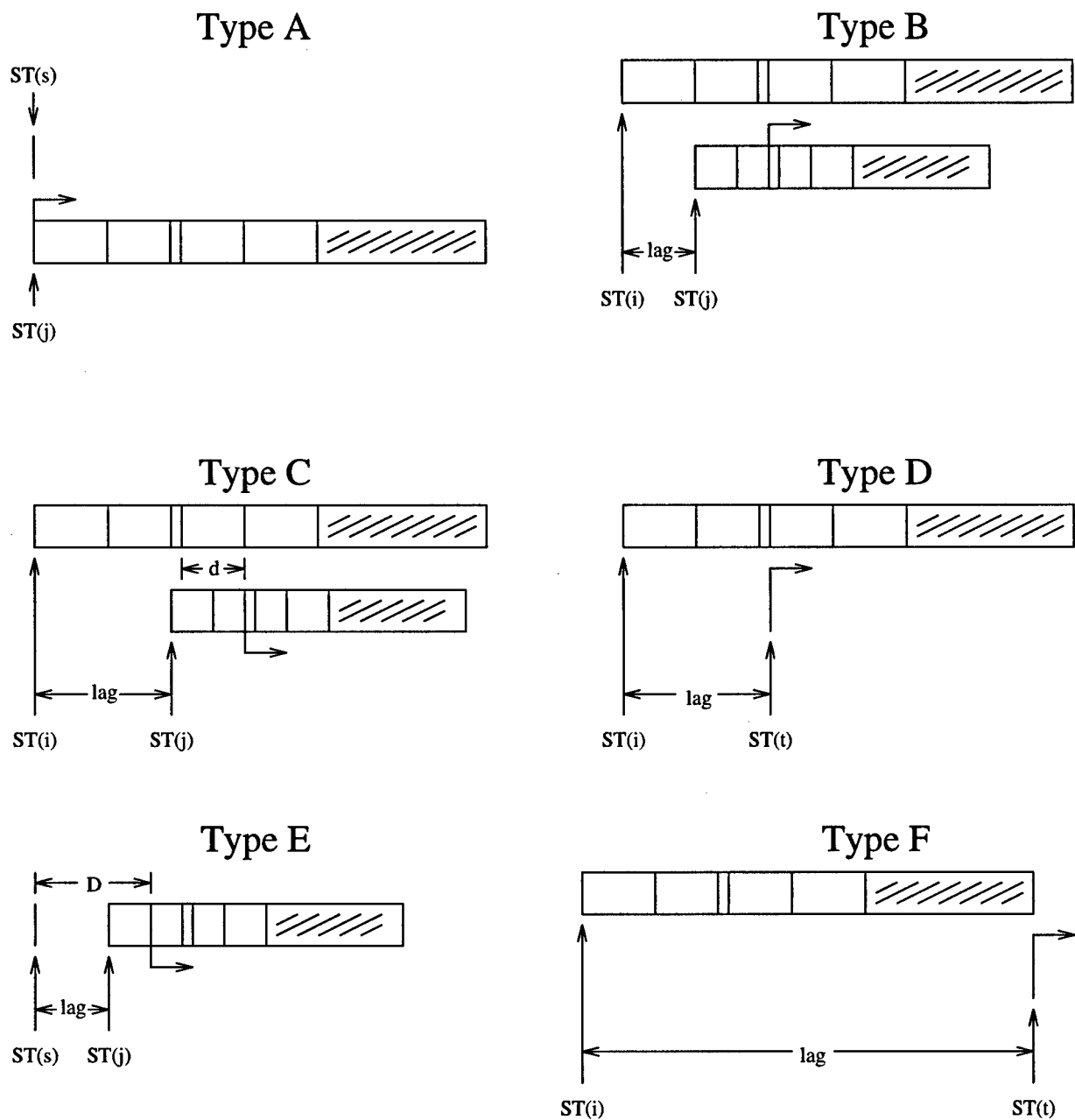


Figure 20 Types of Precedence

first mission is complete. The minimum lag value for this type of precedence condition is given as  $APPR_i + INGR_i + ATT_i - APPR_j - INGR_j$ . Note that since the leg durations vary based on the locations of the targets and bases, the minimum lag value may turn out to be a negative number. If the minimum lag between the start times of predecessor  $i$  and successor  $j$  is -5, the precedence constraint ensures that the start time of  $j$  is no more than 5 minutes earlier than the start time of  $i$ .

Type C precedence is used for situations where mission  $j$  must travel over the target area of mission  $i$  in order to reach its own target. It is assumed that it would be better for mission  $j$  not to pass over the target area of mission  $i$  until mission  $i$  has completed its attack leg. The minimum lag value for this type of precedence is computed in the same manner as the lag for type B precedence, except that additional time ( $d$ ) is added to ensure that mission  $i$  has completed its attack leg before mission  $j$  passes overhead. For the sample problem, the value of  $d$  was fixed to five minutes for all missions using this type of precedence, but any appropriate value could be used for other scenarios. The formula for the computation of the minimum lag value is  $APPR_i + INGR_i + ATT_i + d - APPR_j - INGR_j$ .

Each enemy air defense subproblem in the sample problem includes a zero duration dummy sink activity. This activity becomes the successor of every activity in the subproblem that does not already have a successor. Type D precedence is used to define the precedence relation between this dummy activity and each of its predecessors. The minimum lag time used for this precedence condition is  $APPR_i + INGR_i + ATT_i$  to ensure that the dummy sink activity does not occur until all of its predecessors have completed their attack legs. This dummy sink activity is the link between the subproblem for the air defense targets of a wave and the subproblem for the interdiction targets of the same wave. The missions of an interdiction subproblem may not begin their ingress legs until the dummy sink activity of the air defense subproblem has occurred indicating that all of the air defense targets for that wave have been attacked and it is safe for the interdiction missions to enter enemy airspace.

Type E and F precedence relations are used to enforce precedence conditions with the dummy source and sink activities of the interdiction target subproblems. Type E precedence is used to make sure that the earliest precedence feasible start time for the ingress leg of all successors of the dummy source activity is the same. The minimum lag value for this precedence condition is  $D - APPR_j$

where  $D = \text{MAX}_j(\text{APPR}_j)$ . The combination of type D precedence in the air defense subproblems and type E precedence in the interdiction subproblems enforces the requirement that all air defense targets are attacked before interdiction missions enter enemy airspace. Type F precedence is used to define the end of each wave of attack missions by constraining the dummy sink activity of each interdiction target subproblem to occur only after all interdiction missions are complete. Note that this precedence condition is the same as standard end-to-start activity precedence. The minimum lag value used is simply the duration of the predecessor activity.

Dummy source and sink activities are added to each subproblem bringing the total number of activities in the sample problem to 116. The six types of activity precedence described in Figure 20 are used to formulate all precedence constraints among the 116 activities. The resulting precedence networks are provided in Appendix B. To illustrate the construction of these precedence networks, Figure 21 contains the networks for all of the activities of the third wave of attack missions.

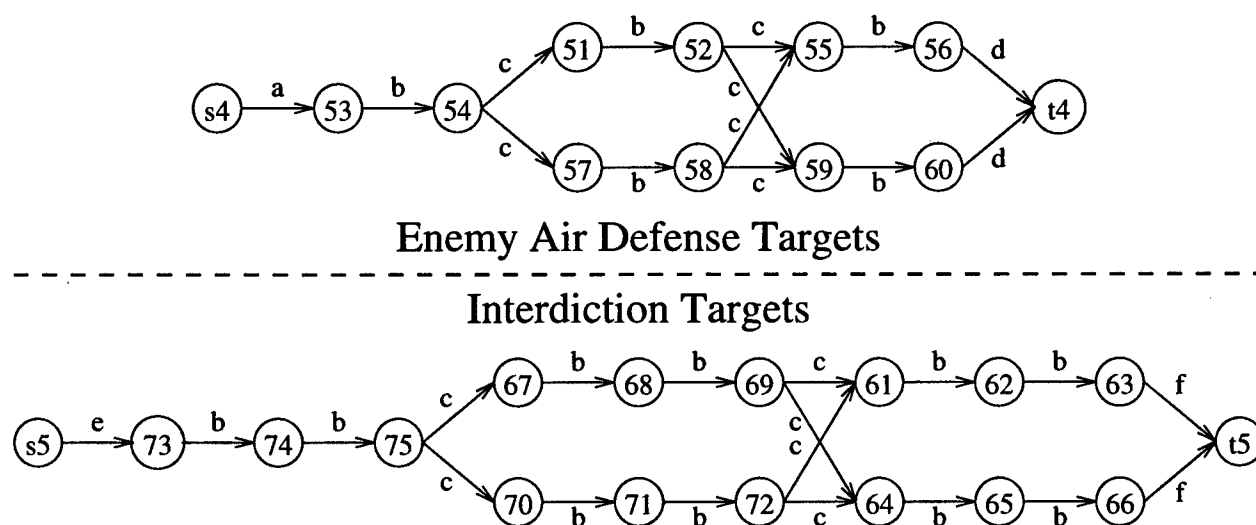


Figure 21 Precedence Networks

The nodes of the precedence networks in Figure 21 correspond to the identification numbers of the targets in the third wave. The dummy source and sink nodes are labeled with  $s_n$  and  $t_n$ , respectively, where  $n$  denotes the number of the subproblem. An arc directed from node  $i$  to node  $j$  in these networks indicates that activity  $j$  is a generalized successor of activity  $i$ . Each arc is labeled with a letter corresponding to the type of precedence relation between the activities connected by that arc. For example, activities  $s_4$  and 53 are connected by an arc indicating type A precedence

since activity  $s_4$  is the dummy source node of the subproblem. The arc between nodes 53 and 54 indicates type B precedence since targets 53 and 54 are at the same target location and must be attacked sequentially. The arc between nodes 54 and 51 indicates type C precedence because it is possible that mission 51 may travel over the target area of mission 54.

Consider a predecessor/successor pair of activities,  $(i, j)$ . If activity  $i$  has  $m$  modes and activity  $j$  has  $n$  modes, there are  $m \times n$  different lag values that may be required to enforce the precedence relation of these two activities. Every potential lag value for every predecessor/successor activity pair has been computed based on the type of precedence relation indicated by the precedence networks and these values are compiled in a series of tables in Appendix B. Table 40 provides an example of how the data is reported in Appendix B.

Table 40 Sample Lag Values

Predecessor		Successor target 51				Successor target 57			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$	$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 54	$m = 1$	0	-14	-8	-25	-6	-6	0	-11
	$m = 2$	9	-5	1	-16	3	3	9	-2
	$m = 3$	5	-9	-3	-20	-1	-1	5	-6
	$m = 4$	22	8	14	-3	16	16	22	11

Table 40 contains all of the possible lag values between activity 54 and its successors, activities 51 and 57. The lag value enforced depends on both the execution mode of the predecessor activity,  $m$ , and the execution mode of the successor activity,  $n$ . The combination of the precedence networks and the tables of lag values provides all the information necessary to formulate the precedence constraints for every predecessor/successor activity pair in the sample problem.

## 7.2 Sample Problem Results

The sample problem was formulated and solved on a 167MHz UltraSPARC using the hybrid variant of the integer program decomposition algorithm developed in this dissertation. The algorithm obtained the 10 best active permutation schedules for the sample problem in approximately 45 minutes. Solving a problem of this size in less than an hour is a significant achievement considering the other options available for obtaining an optimal solution to this type of problem. The only other



exact solution alternatives for this type of problem are to apply a standard IP solver or transform the problem so that Sprecher's algorithm could be applied.

Recall that in Chapter IV a small sample problem was formulated and solved using CPLEX. That problem had approximately 1200 binary variables and 300 constraints and required close to nine hours to solve with CPLEX on a Sun SPARCstation-10. The IP formulation of the case study sample problem would include more than six million binary variables and 80 thousand constraints. This variable count is pessimistic since it does not take into account preprocessing techniques for variable reduction; even so, the the IP formulation of this problem is of daunting proportions. Given the exponential relationship between solution time and the number of variables for standard IP solvers, it may be concluded that this is not a feasible alternative for solving the case study sample problem.

The other alternative is to transform the problem such that generalized precedence constraints are no longer required and use Sprecher's implicit enumeration algorithm to solve the transformed problem. This transformation calls for each of the non-dummy activities to be broken up into its component legs and would require six activities in the transformed problem to take the place of every non-dummy activity in the original problem. The resulting transformed problem would have over 600 total activities. Since the solution times for Sprecher's algorithm become excessive (several hours) when the number of activities gets much over 20 [79], this method is just as impractical for the case study sample problem as a standard IP solver.

The output of the decomposition algorithm used on the sample problem is a list of start times and execution modes for all of the activities. This output for the optimal solution is provided in two forms. The solution information is first displayed in Appendix B as a table of ATO-like mission data. The solution information is then illustrated graphically by a sortie flow chart. Table 41 shows the optimal solution for the third wave of the sample problem as ATO-like mission data.

For the 25 targets that make up the third wave of the sample problem, Table 41 provides the assignment of resources to targets and the important mission scheduling information that would be provided in the mission lines of the ATO in an actual force level planning exercise. Note that the earliest missions in this wave start at or about 1200. It was assumed that the attack waves are temporally discrete and arbitrary start times of 0000, 0600, 1200, and 1800 were assigned to the

waves of the sample problem. To provide a better understanding of how the flow of missions within a wave conform to the generalized precedence constraints specified, the solution information from Table 41 is illustrated graphically as a sortie flow chart in Figure 22.

Table 41 Sample ATO Solution Data

Target	Resources	Takeoff	Attack
51	2 AC-1 from Base A	1224	1306-1307
52	4 AC-2 from Base C	1200	1307-1308
53	2 AC-1 from Base A	1223	1259-1300
54	4 AC-1 from Base A	1224	1300-1301
55	2 AC-2 from Base C	1212	1313-1314
56	4 AC-2 from Base C	1213	1314-1315
57	2 AC-1 from Base B	1218	1306-1307
58	4 AC-1 from Base A	1219	1307-1308
59	2 AC-1 from Base A	1216	1313-1314
60	4 AC-2 from Base C	1219	1314-1315
61	4 AC-1 from Base B	1303	1406-1407
62	4 AC-2 from Base B	1309	1407-1408
63	2 AC-2 from Base B	1310	1408-1409
64	4 AC-1 from Base A	1308	1406-1407
65	4 AC-1 from Base B	1259	1407-1408
66	4 AC-1 from Base A	1310	1408-1409
67	4 AC-2 from Base C	1253	1358-1359
68	2 AC-1 from Base A	1307	1359-1400
69	2 AC-1 from Base B	1301	1400-1401
70	4 AC-1 from Base B	1256	1358-1359
71	4 AC-1 from Base A	1301	1359-1400
72	4 AC-1 from Base B	1258	1400-1401
73	4 AC-2 from Base B	1300	1350-1351
74	2 AC-1 from Base B	1255	1351-1352
75	2 AC-1 from Base A	1256	1352-1353

Each horizontal line on the graph in Figure 22 represents the entire duration, including turn time, of a mission in the third wave of the sample problem. The x's on the horizontal lines mark the beginning, end, and transition points in the duration of each mission. For example, the first x on each line marks the time that the mission takes off from its base, the second x marks the mission's entry into enemy airspace, the the third and fourth x's, grouped closely together on each

line, mark the attack window for the mission. The relative mission timing displayed in Figure 22 can be verified against the precedence networks illustrated previously in Figure 21.

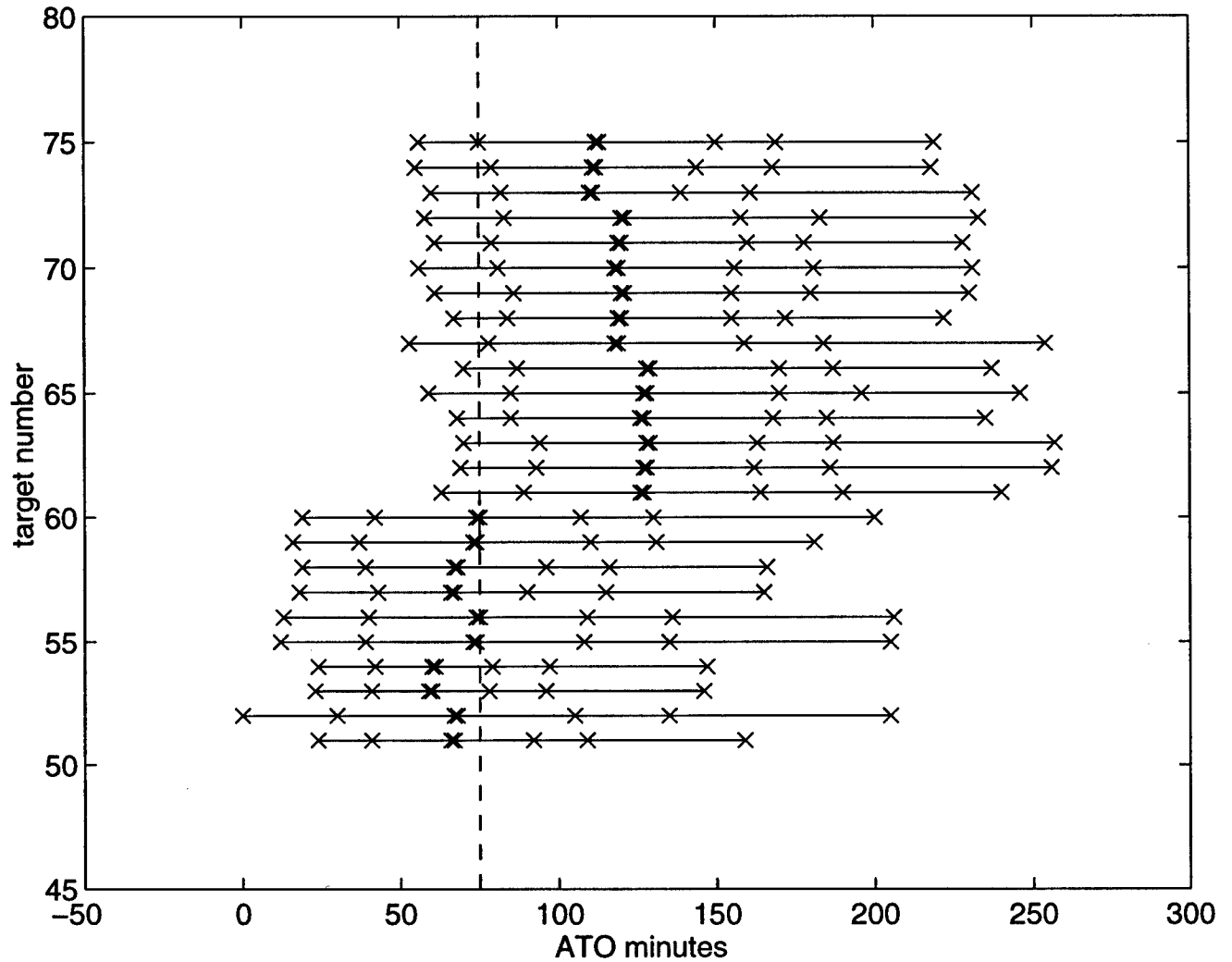


Figure 22 Sortie Flow

The dashed vertical line in Figure 22 represents the precedence requirement between the two subproblems that make up this wave. Missions 51-60 are the enemy air defense targets and form one subproblem while missions 61-75 are the interdiction targets that form the other subproblem. Aside from the precedence conditions among missions in the same subproblem, illustrated by the precedence networks of Figure 21, there is one precedence constraint that binds the two subproblems together into a wave. That precedence condition specifies that all air defense targets are attacked

before any interdiction mission crosses the FLOT. The dashed vertical line on the sortie flow chart illustrates that this precedence condition is met in the optimal solution.

The solution methodology developed in this dissertation includes a technique for obtaining  $k$ -best solution sets. This technique was used to obtain the 10 best active permutation schedules for the case study sample problem. These 10 solutions are summarized in Table 42.

Table 42 Solution Options

Solution #	OBJ	Total Sorties				Peak Usage			
		RES1	RES2	RES3	RES4	RES1	RES2	RES3	RES4
1	631	100	58	52	80	36	22	16	26
2	632	88	60	60	78	36	24	24	26
3	632	104	54	50	82	36	22	14	26
4	634	90	60	60	80	32	18	18	28
5	634	90	58	48	90	32	18	14	28
6	636	96	56	60	78	36	18	22	24
7	638	94	68	44	80	32	24	14	26
8	638	102	54	58	78	36	22	18	24
9	646	92	58	60	76	34	20	22	26
10	647	98	56	60	76	36	18	22	22
Limits		108	72	60	90	36	24	24	36

The solutions reported in Table 42 are listed in order of increasing objective function value. Recall that the objective of the problem is to minimize the span of time that the missions of each wave are in enemy airspace and subject to the threat of enemy air defenses. The objective function values are reported in minutes so the difference between the optimal solution and the 10<sup>th</sup> best solution is 16 minutes of exposure to enemy air defenses. Since the difference among the objective function values of these solutions is relatively small, it is useful to discuss what criteria might be used to choose from a set of near optimal solutions.

One approach to developing selection criteria could be based on the utilization of resources by the various near optimal solutions. Table 42 provides information for each solution that reports both the total number of sorties flown by each unit and the peak aircraft usage for each unit. The total sortie information corresponds to the nonrenewable sortie constraints while the peak usage information is related to the renewable resource constraints that limit the number physical airframes that can be in the air at any given time. The last line of the table provides the limits that represent the right-hand-side values of the resource constraints.

While all of the solutions in Table 42 have good objective function values, the resource utilization information might make one solution more attractive than another. If the optimal solution has one unit providing the maximum possible number of sorties while the other units are underutilized, it may be worth a few extra minutes of exposure to enemy air defenses in order to select a solution with a more equal distribution of sorties among the units. Another reason to choose a solution other than the optimal solution might be peak usage values at or near the maximum for one or more units. If a schedule calls for every aircraft in a unit to be in use simultaneously, there is no room for equipment failure. The risk of missions getting canceled increases. It is apparent that while  $k$ -best solution sets are relatively inexpensive in terms of computation time, they can provide useful solution alternatives for a combat planning problem. Of course, a final plan would be drawn up utilizing the non-quantifiable expertise of the planner. Having a set of baseline alternatives, however, would aid the process.

### *7.3 Summary*

This chapter presented a case study to demonstrate the applicability of the techniques developed in this dissertation research to the force level air campaign planning problem. The sample problem for the case study was generated to provide examples of how decomposition, multiple execution modes, doubly constrained resources, and generalized precedence may be applied to combat mission planning. In addition, the case study presents criteria for discriminating between the set of near optimal solutions provided by the hybrid decomposition algorithm. Chapter VIII summarizes the research conducted in this dissertation, the significant contributions of the research, and the recommendations for future research.

### *VIII. Conclusions and Recommendations*

This chapter presents a summary of the research in this dissertation. The significant contributions of the research are outlined and recommendations for future research are provided.

#### *8.1 Research*

The research accomplished two primary goals. The first goal was to develop an effective solution methodology for a class of resource constrained project scheduling problems with activities having both multiple execution modes and generalized precedence constraints, a problem referred to in this dissertation as the MMGRCPSP. The second goal was to demonstrate the applicability of the MMGRCPSP model to joint campaign planning in general, and specifically, to the problem of the force level planning of combat missions in an air campaign.

The literature offers no specialized solution methods for the MMGRCPSP and general solution methods are computationally impractical for all but the smallest instances of this class of problems. There is, however, research in the literature that proposes an effective specialized algorithm for the class of resource constrained project scheduling problems with activities that have multiple execution modes, but deals with only standard, end-to-start precedence relations, a problem referred to in this dissertation as the MMRCPPSP. This research effort developed an exact algorithm for the MMGRCPSP.

The extended algorithm developed in Chapter IV provides a means for solving MMGRCPSP instances more effectively than any currently available approach. While the algorithm performs well on problems of the magnitude tested in the literature, the problem is NP-complete. The exponential relationship between problem size and solution time causes the solution times required for larger problems to be unreasonably high. To increase the size of problems that can be solved within reasonable time and hardware limitations, the research explores the possibility of exploiting block-angular structure in larger problem instances through the use of decomposition techniques.

The MMGRCPSP formulation developed is an integer program. The only general decomposition approach available in the literature that exploits block-angular structure in integer programs is Sweeney-Murphy decomposition. While Sweeney-Murphy decomposition is theoretically sound, some of the computation involved is cumbersome and time consuming. Sweeney-Murphy decompo-

sition has not been widely applied in the literature due to these limitations. Thus, it was necessary to develop techniques to address several theoretical and operational aspects of the Sweeney-Murphy algorithm in order to effectively implement a decomposition approach to the MMGRCPSP.

The two major difficulties with implementing Sweeney-Murphy decomposition are finding *proxy dual variables* for the linking constraints in the master problem and generating *k*-best solution sets for the subproblems. The research developed a method of parametric analysis to estimate the marginal benefit of increasing the right-hand-side of each linking constraint by one unit. These marginal benefit values were shown to be effective *proxy dual variables* for an implementation of Sweeney-Murphy decomposition.

To address the issue of *k*-best subproblem solution sets, the research in this dissertation developed an enhanced implicit enumeration strategy that obtains the *k*-best solutions to a problem in only slightly more time than it takes to find a single optimal solution for that problem. The strategy proposed serves the original purpose of finding *k*-best subproblem solution sets for Sweeney-Murphy decomposition, but it is a general approach that can be applied to any implicit enumeration algorithm. In fact, the *k*-best strategy was applied to the solution algorithm for the Sweeney-Murphy master problem to provide a set of near optimal solution alternatives for any problem to which the decomposition algorithm is applied.

The combination of the specialized MMGRCPSP solution algorithm and the integer program decomposition technique developed provides an effective method for solving larger problem instances than previously possible. However, the problem is still NP-complete. To provide an alternative for problems that are too large for the exact algorithm to solve in a reasonable amount of time, a heuristic solution alternative, evolutionary algorithms, was explored. As with the exact algorithms, there are no EA implementations found in the literature for the MMGRCPSP, but there are EA implementations for the MMRCPPSP.

The research extended EA procedures proposed in the literature for the MMRCPPSP to develop an EA implementation for the MMGRCPSP. The MMGRCPSP EA implementation was then tested against the decomposition method in order to show that the solution times of the EA increase at a significantly slower rate with respect to problem size than the solution times of the decomposition method. When solution time is a more pressing concern than solution accuracy, the EA offers a less

computationally intensive alternative to the exact algorithm. In addition to providing a heuristic solution alternative, the EA was used to develop a more efficient method of estimating marginal benefit for Sweeney-Murphy decomposition. The hybrid decomposition approach that used the EA to estimate marginal benefit was shown to significantly improve solution time for larger problem instances.

The hybrid decomposition approach is the culmination of the goal to develop an effective solution methodology for the MMGRCPSP. In pursuit of the second dissertation goal, the hybrid approach was applied in a case study in order to demonstrate the applicability of the MMGRCPSP model to campaign planning. The case study discussed how a MMGRCPSP model would be formulated from the planning data available for an operational scenario. The sample problem used, however, was a contrived example. The purpose of the example was to demonstrate a modeling technique, not to advocate any particular combat planning strategy.

The first part of the case study involved the generation of the sample problem. This part of the study explained the various bits of information that are necessary to completely describe the activities, resources, and precedence relations that make up the MMGRCPSP model formulation for a combat planning problem instance. While it is relatively straight forward to define activities and resources, there are countless alternatives for restricting the timing of the activities with generalized precedence constraints and for decomposing the problem into subproblems.

Several options were presented for decomposing a combat planning problem and two of these options were illustrated in the sample problem. The sample problem was decomposed into waves of attack missions and each attack wave was further decomposed by target type. Six different variations of generalized precedence were applied to enforce various mission timing requirements. These mission timing requirements were used either to efficiently sequence missions around the target areas or to keep certain deeper missions out of enemy airspace until enemy air defense could be suppressed. It was stressed that the precedence relations demonstrated in the sample problems were merely examples and that generalized precedence constraints are flexible enough to enforce virtually any mission timing requirement.

The final effort in the case study was to discuss methods for exploiting the  $k$ -best solutions provided by the hybrid decomposition algorithm. The hybrid algorithm was used to obtain the 10



best solutions to the sample combat planning problem. It was observed that while the objective function values of these 10 solutions were nearly identical, the utilization of resources varied greatly. Several criteria were proposed to aid in the process of selecting from a set of near optimal solutions.

## *8.2 Contributions*

The contributions of this research begin with the formulation of the MMGRCPSP model for combat planning. This problem formulation not only represents a new approach to combat planning, but it is also an original contribution to the field of resource constrained project scheduling. This field of research has seen formulations for multiple activity execution modes, and formulations for generalized precedence constraints, but the model is the first to incorporate both.

Aside from being the first of its kind, the formulation of generalized precedence for multi-modal activities developed is attractive for its efficient use of variables and constraints. This formulation has the ability to enforce a large set of possible precedence conditions with a relatively small number of constraints and without any additional variables to enforce the logic that selectively enforces the appropriate precedence condition. In a model that quickly becomes unwieldy as problem size grows, the efficient precedence constraints are very much appreciated.

The new problem formulation is a significant contribution, but it is of limited use without specialized algorithms that can solve large problem instances effectively. The implicit enumeration algorithm and the EA approach developed are what make the MMGRCPSP model formulation useful. The former offers an exact solution approach for small to medium sized problems that require an optimal solution while the latter provides a heuristic alternative for problems too large to solve with the exact approach in a realistic amount of time. These solution methodologies are original approaches to a new class of problems and as such represent significant contributions to the field of resource constrained project scheduling.

The efforts at integer program decomposition presented in this dissertation enhance an existing decomposition approach and apply it to a new problem, the MMGRCPSP. These efforts develop several useful new techniques that make the decomposition approach more effective. The general  $k$ -best solutions strategy for implicit enumeration is an original contribution that not only increases

the effectiveness of the decomposition effort, but can be used to enhance any implicit enumeration algorithm by providing multiple near optimal solution alternatives for every problem solved.

In addition to the  $k$ -best solutions strategy, the decomposition efforts also resulted in the development of efficient methods for estimating *proxy dual variables* in large integer programs. In Chapter V it was shown that it is not possible to estimate *proxy dual variables* for the MMGRCPSP by standard LP relaxation techniques. Without the marginal benefit estimation procedures developed in this dissertation research, there would be no way to derive meaningful Lagrangian multipliers to use in the Sweeney-Murphy subproblem formulations and it would be unlikely that an efficient decomposition approach could be implemented for the MMGRCPSP. In addition to the increased efficiency these marginal benefit values bring to the decomposition efforts, they have applications in the sensitivity analysis of any integer program problem.

The final contribution of this dissertation research is the case study. The MMGRCPSP formulation represents a new class of project scheduling problems. This research offers several effective solution approaches to this new problem area, but it is the case study that demonstrates that the model and solution approaches have applications to real world problems. In generating and solving a combat planning scenario of approximately 300 combat sorties, the case study illustrates the applicability of the methodology. The case study also applies multiple variations of the general modeling approach in order to demonstrate the flexibility of the model.

### 8.3 Recommendations

The following recommendations are made for extending the research:

1. The implicit enumeration algorithm was based on an algorithm for the MMRCPPSP. Not all of the bounding strategies in the original algorithm were applicable to the MMGRCPSP. Those bounding strategies that were applicable were used directly and an effort was made to find suitable alternatives for non-applicable strategies wherever possible. The resulting algorithm proved to be roughly as effective as the MMRCPPSP algorithm, but there is room for improvement in the bounding strategies. Additional effective bounds might be found to speed up the solution times of the algorithm and increase the size of problems that can be solved in realistic amounts of time.

2. In Chapter VI, Hartmann's EA for the MMGRCPSP [35] is extended to address the generalized precedence constraints of the MMGRCPSP. To accommodate generalized precedence constraints, it was necessary to develop a modified fitness function. However, Hartmann's genetic representation of individuals and genetic operators were used directly in the extended EA. The extended approach represents one possible EA for the MMGRCPSP, but it is not necessarily the most effective evolutionary approach. A survey of the genetic representations of individuals and the genetic operators available in the literature may lead to a more effective EA.
3. Experiments were conducted to determine the optimal EA parameter settings for the size and type of problems that were used for testing. These settings optimized the results of the EA based on a comparison to optimal solutions found by the implicit enumeration algorithm. The EA literature has shown that the use of island populations can improve the performance of some EA implementations. The EA implementation could be extended to include island populations in order to determine the extent to which the performance of the EA could be improved.
4. In this dissertation, it is demonstrated that the choice of multiplier values in Sweeney-Murphy decomposition can have a large effect on computation time. Sweeney and Murphy explain how initial optimal multipliers are found, but in most cases it is not realistic to compute the multiplier values they describe. The decomposition implementation obtained good results using marginal benefit values as Lagrangian multipliers, but it may be possible to find alternate values whose derivation requires less computational effort. The problem generator ProGen requires settings for resource factor and resource strength in order to generate the test problems and so these parameters are readily available for every test problem used in this research. These parameters measure the demand for, and availability of, every resource in the test problems. It may be possible to estimate a value similar to marginal benefit using these parameters. If so, it would be possible to derive meaningful Lagrangian multipliers for the decomposition with virtually no computation time. This approach has potential beyond the possibility of reducing the time required to compute multiplier values. It is possible that the new multiplier values may be better measures of the value of project resources and may lead

to smaller  $k$ -best solution sets, fewer Sweeney-Murphy iterations, and a reduction in overall solution times.

5. In the decomposition effort, a specialized algorithm was developed to solve the Sweeney-Murphy master problem. This specialized algorithm exploited structure to solve the master problem much faster than a standard IP solver could. However, during the execution of the decomposition algorithm, each sequential solution of the master problem implicitly enumerates every solution in the solution space of the previous master problem. This is a waste of computation time and it should be possible to modify the master problem implicit enumeration algorithm such that solutions pruned from the search tree of one master problem are automatically pruned from the solution tree of every additional master problem. This would increase the efficiency of the master problem solution algorithm and improve the solution time of the overall decomposition algorithm.
6. The Sweeney-Murphy decomposition algorithm is the only general decomposition algorithm available in the literature for block-angular integer programs. This dissertation research overcomes some of the cumbersome aspects of this algorithm and implements an effective decomposition approach. Another option is to develop a new decomposition approach for block-angular integer programs. The technique of decomposition by right-hand-side allocation is a decomposition method in the literature for block-angular linear programs. Initial investigations suggest that it may be possible to extend this technique to block-angular integer programs. The major difficulty with such an extension would be the requirement for finding the dual variables of the subproblems. The discussion of *proxy dual variables* in this dissertation may provide the means for extending this technique to integer programs.
7. Every solution approach developed in this dissertation applies some combination of implicit enumeration, evolutionary algorithms, and decomposition techniques. For all three of these categories, there are examples in the literature of how solution times may be reduced through parallelization. The core of the research has been finding ways to solve large instances of difficult problems as efficiently as possible. Parallelizing some or all of the techniques for solving MMGRCPSP instances would be a natural extension. A parallel version of these algorithms would also allow hybrid approaches to be explored more thoroughly. This would include, but

not be limited to, exchanging solutions between the implicit enumeration algorithm and the EA while they are running in parallel.

8. The case study in this dissertation used a contrived example to demonstrate the applicability of the approach. It would be useful to go one step further and demonstrate the approach on a real world scenario. There are many force level planning tools in the operational community from which most of the data required to formulate such a scenario could be automatically extracted. An interface between the algorithms of this dissertation and a force level planning tool would facilitate additional case studies involving real data. This interface could bridge the gap between theoretical dissertation research and useful operational planning tools. In addition, it would be useful to formulate a MMGRCPSP for an operational scenario and compare the solution provided by the algorithms in this dissertation with a solution developed by combat planning experts.

#### 8.4 *Summary*

This research formulated and solved a new class of project scheduling problems, the MMGRCPSP, with applications to both military and civilian planning. It was shown that the solution space for this class of problems may be reduced in order to improve the effectiveness of both optimal and heuristic solution methodologies. In addition, a general method for extending implicit enumeration algorithms to obtain  $k$ -best solution sets was developed. The reduced solution space and the general  $k$ -best solutions methodology were exploited to develop several efficient solution approaches for the MMGRCPSP; an implicit enumeration algorithm, a decomposition approach, an evolutionary algorithm, and a hybrid decomposition approach. The applicability and flexibility of the methodology was demonstrated with a case study that focused on the force level planning of combat missions for an air campaign.

### Appendix A. Setting $k_i$ Values

When choosing a  $k_i$  selection strategy, the primary concern is how the chosen strategy affects the overall solution time for the decomposition algorithm. Assume that, for subproblem  $i$  of some problem instance,  $k_i^*$  is the minimum value of  $k_i$  for which the Sweeney-Murphy optimality criteria will be met. Suppose that  $\bar{k}_i$  is the  $k_i$  value used for one of the Sweeney-Murphy iterations for this problem. Let  $\Delta = \bar{k}_i - k_i^*$ . The ideal case is when  $\Delta = 0$ . In this case, the optimality criteria are met with the smallest possible number of solutions to subproblem  $i$  included in the master problem.

If  $\Delta > 0$ , the optimality criteria are met, but the master problem was larger than necessary. Regardless of the magnitude of  $\Delta$ , the exponential nature of the master problem solution algorithm could result in a significant difference between the solution time required to solve a master problem including  $k_i^*$  subproblem solutions and the solution time required to solve one that includes  $\bar{k}_i$  subproblem solutions. If the master problem solution algorithm accounts for the largest portion of the overall solution time, a good strategy might be to increase  $k_i$  by small increments from one iteration to the next. This strategy would insure that when the optimality criteria are met,  $\Delta$  will be small and the amount of unnecessary master problem computation time is minimized.

If  $\Delta < 0$ , the master problem did not include enough subproblem solutions for optimality criteria to be met and further iterations are necessary. To accomplish a new iteration,  $k_i$  is increased and subproblem  $i$  and any other subproblems that did not meet the optimality criteria are re-solved. If the subproblem solution algorithm accounts for the largest portion of the overall solution time, every iteration, regardless of the value of  $k_i$ , is computationally expensive. In this case, a good  $k_i$  selection strategy might be one that increases  $k_i$  by large increments from one iteration to the next in order to minimize the total number of iterations before the optimality criteria are met.

When considering the merits of various  $k_i$  selection strategies, there is a tradeoff between the goal of minimizing the value of  $\Delta$  and the goal of minimizing the number of Sweeney-Murphy iterations. The choice between these two goals may be decided based on which Sweeney-Murphy component, the subproblem solution algorithm or the master problem solution algorithm, accounts for the largest portion of the overall solution time, which in turn depends on the size and complexity of the subproblems. If the subproblems are *small*, the Sweeney-Murphy iterations do not become computationally expensive until the value of  $k_i$  grows to the point where the exponential nature of

the master problem solution algorithm becomes apparent and minimizing the value of  $\Delta$  is the most important goal. If the subproblems are *large*, every Sweeney-Murphy iteration is computationally expensive and minimizing the number of iterations is the most important goal.

Empirical tests were conducted in order to show the breakpoint, in terms of the number of activities, between *small* and *large* subproblems. Each problem in the set of test problems has three subproblems and all of the subproblems for a given problem instance have either 10, 15, 20, or 25 activities. The test set contains 10 problem instances for each level of subproblem activities. The test problems are generated with ProGen using the same complexity parameters used for the test problems in Chapter V.

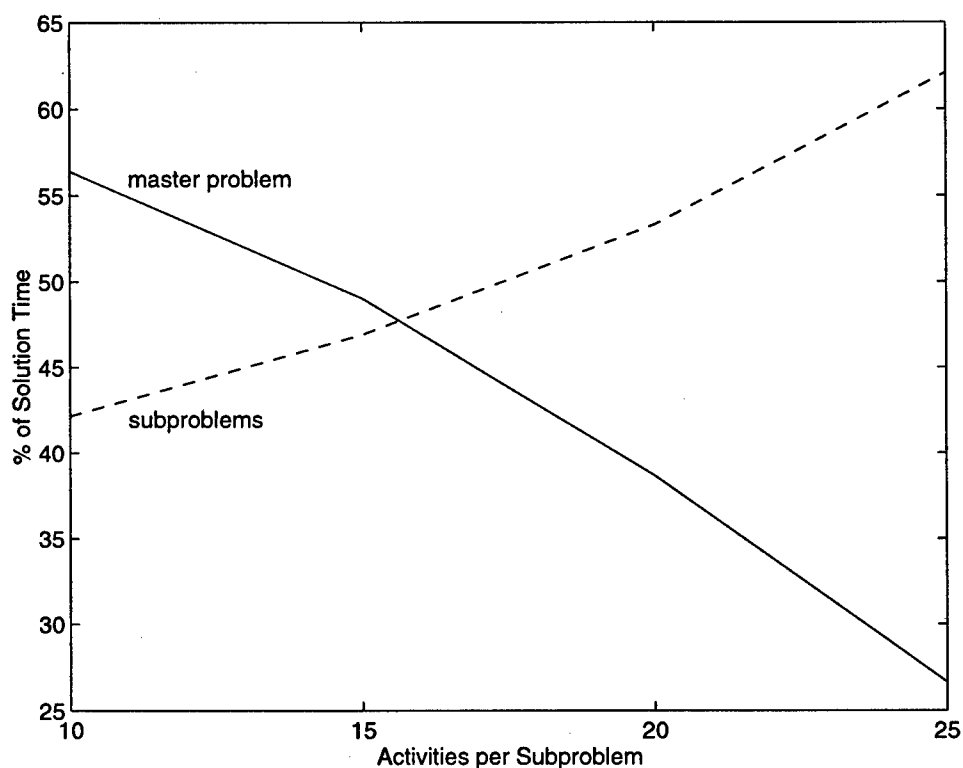


Figure 23 Distribution of Solution Time

For each level of subproblem activities, the graph in Figure 23 shows the average percent of total solution time spent on solving the master problem versus the average percent of total solution time spent on solving the subproblems. The point at which the two lines intersect on the graph provides an indication of the subproblem size at which the subproblem solution algorithm

begins to dominate the overall solution time for the decomposition algorithm. The breakpoint is at approximately 15 activities per subproblem.

For the problems specified in this dissertation, the breakpoint illustrated in Figure 23 allows tailoring of the  $k_i$  selection strategy for a given subproblem solution size. If the decomposition algorithm is fine tuned for problems with subproblems smaller than 15 activities, an appropriate strategy might be to increase  $k_i$  by a small constant at every iteration in order to minimize  $\Delta$ . If the decomposition algorithm is fine tuned for subproblems with more than 15 activities, the best strategy might be to increase  $k_i$  by a large constant or use a doubling strategy in order to minimize the total number of Sweeney-Murphy iterations. If the algorithm is fine tuned for problems with both *small* and *large* subproblems, the most appropriate  $k_i$  selection strategy might be one that offers a compromise between minimizing  $\Delta$  and minimizing the number of iterations. Since the test sets used in Chapter V include subproblems on either side of the breakpoint, experiments showed that a compromise strategy was necessary. The compromise strategy used a small initial  $k_i$  value ( $k_i = 10$ ), doubled  $k_i$  for early iterations until  $k_i$  exceeded 100, and then added 50 to  $k_i$  for every additional iteration.

To apply the general  $k_i$  selection guidelines outlined in this appendix for some other class of decomposable integer program problems, the first step is to find the breakpoint, in terms of subproblem solution size, at which the bulk of the overall computation time switches from the master problem solution algorithm to the subproblem solution algorithm. This breakpoint could be obtained from the results of empirical tests or through theoretical complexity analysis. Once a breakpoint has been found, the  $k_i$  selection strategy may be fine tuned based on the characteristics of the problems to which the algorithm is likely to be applied. When subproblems are smaller than the breakpoint size, the  $k_i$  strategy should take smaller steps between iterations to minimize  $\Delta$ . For subproblems that exceed the breakpoint size, the  $k_i$  strategy should take larger steps to minimize the number of iterations. If the subproblems fall on both sides of the breakpoint, the most appropriate strategy might be a compromise between minimizing  $\Delta$  and minimizing iterations.



## *Appendix B. Case Study Problem Data*

This appendix contains all of the data necessary to formulate the sample problem used in the case study. Chapter VII contains a subset of the data presented in each section of this appendix. The case study scenario contains 100 targets and any solution generated for this scenario includes 200-400 sorties.

### *B.1 Target Nomination List*

This section contains the target nomination list (TNL) for the case study sample problem. The first five columns of data in the TNL provide information about a target location, specifically the identification number from the map, the (x,y) coordinates of the location, and the type and number of targets located at those coordinates. The next two columns offer information about each of the targets at a given location including a target identification number and the number of possible execution modes for that target. The last two columns give the details for every execution mode which consist of a mode identification number and the number, type, and location of the necessary resources.

LOC	X	Y	Target Type	NTAR	TAR	NMOD	MOD	Num/Type Resources
1	4	60	Air Defense	2	1	2	1	4 AC-2 from Base B
							2	4 AC-2 from Base C
							1	4 AC-2 from Base B
							2	4 AC-2 from Base C
2	12	52	Air Defense	2	3	4	1	4 AC-1 from Base A
							2	4 AC-1 from Base B
							3	4 AC-2 from Base B
							4	4 AC-2 from Base C
							1	4 AC-2 from Base B
							2	4 AC-2 from Base C
3	20	56	Air Defense	2	5	4	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
							3	2 AC-2 from Base B
							4	2 AC-2 from Base C
							1	4 AC-1 from Base A
							2	4 AC-1 from Base B
4	28	53	Air Defense	2	7	4	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
							3	2 AC-2 from Base B
							4	2 AC-2 from Base C
							1	2 AC-1 from Base A
							2	2 AC-1 from Base B
							3	2 AC-2 from Base B
							4	2 AC-2 from Base C

LOC	X	Y	Target Type	NTAR	TAR	NMOD	MOD	Num/Type Resources
5	31	59	Air Defense	2	9	2	1	4 AC-1 from Base A
							2	4 AC-1 from Base B
					10	4	1	4 AC-1 from Base A
							2	4 AC-1 from Base B
							3	4 AC-2 from Base B
							4	4 AC-2 from Base C
6	8	72	Interdiction	3	11	2	1	2 AC-2 from Base B
							2	2 AC-2 from Base C
					12	4	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
							3	2 AC-2 from Base B
							4	2 AC-2 from Base C
					13	2	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
7	12	64	Interdiction	3	14	2	1	4 AC-1 from Base A
							2	4 AC-1 from Base B
					15	4	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
							3	4 AC-2 from Base B
							4	4 AC-2 from Base C
					16	4	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
							3	2 AC-2 from Base B
							4	2 AC-2 from Base C

LOC	X	Y	Target Type	NTAR	TAR	NMOD	MOD	Num/Type Resources
8	16	76	Interdiction	3	17	2	1	4 AC-2 from Base B
							2	4 AC-2 from Base C
					18	2	1	2 AC-2 from Base B
							2	2 AC-2 from Base C
					19	4	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
							3	4 AC-2 from Base B
							4	4 AC-2 from Base C
9	19	70	Interdiction	3	20	2	1	4 AC-1 from Base A
							2	4 AC-1 from Base B
					21	2	1	2 AC-2 from Base B
							2	2 AC-2 from Base C
					22	2	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
10	26	66	Interdiction	3	23	4	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
							3	4 AC-2 from Base B
							4	4 AC-2 from Base C
					24	2	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
					25	4	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
							3	2 AC-2 from Base B
							4	2 AC-2 from Base C

LOC	X	Y	Target Type	NTAR	TAR	NMOD	MOD	Num/Type Resources
11	68	52	Air Defense	2	26	4	1	4 AC-1 from Base A
							2	4 AC-1 from Base B
							3	4 AC-2 from Base B
							4	4 AC-2 from Base C
							1	2 AC-2 from Base B
							2	2 AC-2 from Base C
12	71	57	Air Defense	2	28	4	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
							3	2 AC-2 from Base B
							4	2 AC-2 from Base C
							1	2 AC-2 from Base B
							2	2 AC-2 from Base C
13	78	54	Air Defense	2	30	2	1	2 AC-2 from Base B
							2	2 AC-2 from Base C
							1	4 AC-1 from Base A
							2	4 AC-1 from Base B
							3	4 AC-2 from Base B
							4	4 AC-2 from Base C
14	86	59	Air Defense	2	32	4	1	4 AC-1 from Base A
							2	4 AC-1 from Base B
							3	2 AC-2 from Base B
							4	2 AC-2 from Base C
							1	2 AC-2 from Base B
							2	2 AC-2 from Base C

LOC	X	Y	Target Type	NTAR	TAR	NMOD	MOD	Num/Type Resources
15	95	56	Air Defense	2	34	2	1	2 AC-2 from Base B
							2	2 AC-2 from Base C
					35	4	1	4 AC-1 from Base A
							2	4 AC-1 from Base B
							3	4 AC-2 from Base B
							4	4 AC-2 from Base C
16	72	71	Interdiction	3	36	4	1	4 AC-1 from Base A
							2	4 AC-1 from Base B
							3	4 AC-2 from Base B
							4	4 AC-2 from Base C
					37	2	1	4 AC-2 from Base B
							2	4 AC-2 from Base C
					38	2	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
17	75	65	Interdiction	3	39	2	1	4 AC-2 from Base B
							2	4 AC-2 from Base C
					40	2	1	4 AC-1 from Base A
							2	4 AC-1 from Base B
					41	2	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
18	81	76	Interdiction	3	42	2	1	2 AC-2 from Base B
							2	2 AC-2 from Base C
					43	2	1	2 AC-2 from Base B
							2	2 AC-2 from Base C
					44	2	1	2 AC-2 from Base B
							2	2 AC-2 from Base C

LOC	X	Y	Target Type	NTAR	TAR	NMOD	MOD	Num/Type Resources
19	85	67	Interdiction	3	45	2	1	4 AC-1 from Base A
							2	4 AC-1 from Base B
					46	2	1	4 AC-2 from Base B
							2	4 AC-2 from Base C
					47	4	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
							3	2 AC-2 from Base B
							4	2 AC-2 from Base C
20	92	68	Interdiction	3	48	2	1	4 AC-2 from Base B
							2	4 AC-2 from Base C
					49	4	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
							3	2 AC-2 from Base B
							4	2 AC-2 from Base C
					50	4	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
							3	2 AC-2 from Base B
							4	2 AC-2 from Base C
21	36	72	Air Defense	2	51	4	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
							3	4 AC-2 from Base B
							4	4 AC-2 from Base C
					52	4	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
							3	4 AC-2 from Base B
							4	4 AC-2 from Base C

LOC	X	Y	Target Type	NTAR	TAR	NMOD	MOD	Num/Type Resources
22	42	64	Air Defense	2	53	2	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
					54	4	1	4 AC-1 from Base A
							2	4 AC-1 from Base B
							3	2 AC-2 from Base B
							4	2 AC-2 from Base C
23	45	73	Air Defense	2	55	2	1	2 AC-2 from Base B
							2	2 AC-2 from Base C
					56	4	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
							3	4 AC-2 from Base B
							4	4 AC-2 from Base C
24	54	70	Air Defense	2	57	4	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
							3	2 AC-2 from Base B
							4	2 AC-2 from Base C
					58	2	1	4 AC-1 from Base A
							2	4 AC-1 from Base B
25	60	76	Air Defense	2	59	4	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
							3	4 AC-2 from Base B
							4	4 AC-2 from Base C
					60	2	1	4 AC-2 from Base B
							2	4 AC-2 from Base C



LOC	X	Y	Target Type	NTAR	TAR	NMOD	MOD	Num/Type Resources
26	38	82	Interdiction	3	61	2	1	4 AC-1 from Base A
							2	4 AC-1 from Base B
					62	4	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
							3	4 AC-2 from Base B
							4	4 AC-2 from Base C
					63	2	1	2 AC-2 from Base B
							2	2 AC-2 from Base C
27	40	87	Interdiction	3	64	2	1	4 AC-1 from Base A
							2	4 AC-1 from Base B
					65	4	1	4 AC-1 from Base A
							2	4 AC-1 from Base B
							3	4 AC-2 from Base B
							4	4 AC-2 from Base C
					66	2	1	4 AC-1 from Base A
							2	4 AC-1 from Base B
28	45	80	Interdiction	3	67	2	1	4 AC-2 from Base B
							2	4 AC-2 from Base C
					68	2	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
					69	4	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
							3	4 AC-2 from Base B
							4	4 AC-2 from Base C

LOC	X	Y	Target Type	NTAR	TAR	NMOD	MOD	Num/Type Resources
29	50	84	Interdiction	3	70	4	1	4 AC-1 from Base A
							2	4 AC-1 from Base B
							3	2 AC-2 from Base B
							4	2 AC-2 from Base C
					71	4	1	4 AC-1 from Base A
							2	4 AC-1 from Base B
							3	4 AC-2 from Base B
							4	4 AC-2 from Base C
					72	4	1	4 AC-1 from Base A
							2	4 AC-1 from Base B
							3	4 AC-2 from Base B
							4	4 AC-2 from Base C
30	54	79	Interdiction	3	73	4	1	4 AC-1 from Base A
							2	4 AC-1 from Base B
							3	4 AC-2 from Base B
							4	4 AC-2 from Base C
					74	4	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
							3	4 AC-2 from Base B
							4	4 AC-2 from Base C
					75	4	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
							3	2 AC-2 from Base B
							4	2 AC-2 from Base C

LOC	X	Y	Target Type	NTAR	TAR	NMOD	MOD	Num/Type Resources		
31	100	68	Air Defense	2	76	2	1	2 AC-2 from Base B		
							2	2 AC-2 from Base C		
					77	4	1	2 AC-1 from Base A		
							2	2 AC-1 from Base B		
							3	4 AC-2 from Base B		
							4	4 AC-2 from Base C		
					78	4	1	2 AC-1 from Base A		
							2	2 AC-1 from Base B		
32	104	71	Air Defense	2			3	2 AC-2 from Base B		
							4	2 AC-2 from Base C		
				79	4	1	4 AC-1 from Base A			
						2	4 AC-1 from Base B			
						3	2 AC-2 from Base B			
						4	2 AC-2 from Base C			
				80	4	1	4 AC-1 from Base A			
						2	4 AC-1 from Base B			
33	109	64	Air Defense			2			3	2 AC-2 from Base B
									4	2 AC-2 from Base C
				81	2	1	4 AC-1 from Base A			
						2	4 AC-1 from Base B			
				82	2	1	2 AC-2 from Base B			
						2	2 AC-2 from Base C			
				83	2	1	4 AC-1 from Base A			
						2	4 AC-1 from Base B			
34	114	66	Air Defense			2				

LOC	X	Y	Target Type	NTAR	TAR	NMOD	MOD	Num/Type Resources
35	122	70	Air Defense	2	84	4	1	4 AC-1 from Base A
							2	4 AC-1 from Base B
							3	2 AC-2 from Base B
							4	2 AC-2 from Base C
					85	2	1	4 AC-2 from Base B
							2	4 AC-2 from Base C
36	103	78	Interdiction	3	86	2	1	4 AC-2 from Base B
							2	4 AC-2 from Base C
					87	4	1	4 AC-1 from Base A
							2	4 AC-1 from Base B
							3	2 AC-2 from Base B
							4	2 AC-2 from Base C
					88	4	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
							3	2 AC-2 from Base B
							4	2 AC-2 from Base C
					89	2	1	4 AC-2 from Base B
							2	4 AC-2 from Base C
					90	2	1	2 AC-2 from Base B
							2	2 AC-2 from Base C
					91	2	1	4 AC-2 from Base B
							2	4 AC-2 from Base C

LOC	X	Y	Target Type	NTAR	TAR	NMOD	MOD	Num/Type Resources
38	114	74	Interdiction	3	92	2	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
					93	4	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
							3	2 AC-2 from Base B
							4	2 AC-2 from Base C
					94	2	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
39	117	80	Interdiction	3	95	4	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
							3	2 AC-2 from Base B
							4	2 AC-2 from Base C
					96	4	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
							3	2 AC-2 from Base B
							4	2 AC-2 from Base C
					97	4	1	2 AC-1 from Base A
							2	2 AC-1 from Base B
							3	4 AC-2 from Base B
							4	4 AC-2 from Base C

LOC	X	Y	Target Type	NTAR	TAR	NMOD	MOD	Num/Type Resources
40	122	84	Interdiction	3	98	4	1	4 AC-1 from Base A
							2	4 AC-1 from Base B
							3	4 AC-2 from Base B
							4	4 AC-2 from Base C
					99	4	1	4 AC-1 from Base A
							2	4 AC-1 from Base B
							3	2 AC-2 from Base B
							4	2 AC-2 from Base C
					100	2	1	4 AC-2 from Base B
							2	4 AC-2 from Base C

## B.2 Available Assets

This section contains information on the assets available for the case study scenario. The data includes the (x,y) coordinates of the base at which each unit is located. In addition, the data provides some information about the aircraft that make up the units. The NAC column gives the number of aircraft in the unit, the TR column gives the turn rate, or number of sorties per day per aircraft, and the NSOR column gives the total number of sorties each unit can perform in a given ATO day ( $NAC \times TR = NSOR$ ). The TT column gives the aircraft turn time which specifies the amount of time it takes to service aircraft after they have completed a mission. This value is given in minutes. The final column of data contains a nominal airspeed value for each aircraft type and is given in distance units per minute. This value is set to make the average in-flight time for missions in the sample problem equal to approximately 2 hours.

Unit	Type/Base	X	Y	NAC	TR	NSOR	TT	NAS
1	AC-1 from Base A	24	32	36	3.0	108	50	1.0
2	AC-1 from Base B	64	24	24	3.0	72	50	1.0
3	AC-2 from Base B	64	24	24	2.5	60	70	1.1
4	AC-2 from Base C	96	28	36	2.5	90	70	1.1

### B.3 Mission Component Durations

This section contains information regarding activity durations for the case study sample problem. All duration values are reported in minutes. Note that the last column of data is the complete mission duration for a given activity execution mode. By design, the average of the values reported in this column are approximately 180 minutes or three hours.

LOC	TAR	MOD	APP	ING	ATT	EGR	RET	TUR	SUM
1	1	1	42	21	1	21	42	70	197
		2	55	33	1	33	55	70	247
	2	1	42	21	1	21	42	70	197
		2	55	33	1	33	55	70	247
2	3	1	19	5	1	5	19	50	99
		2	51	8	1	8	51	50	169
		3	46	8	1	8	46	70	179
		4	66	13	1	13	66	70	229
	4	1	46	8	1	8	46	70	179
		2	66	13	1	13	66	70	229
3	5	1	16	8	1	8	16	50	99
		2	41	14	1	14	41	50	161
		3	37	12	1	12	37	70	169
		4	52	21	1	21	52	70	217
	6	1	16	8	1	8	16	50	99
		2	41	14	1	14	41	50	161
4	7	1	16	5	1	5	16	50	93
		2	38	8	1	8	38	50	143
		3	34	7	1	7	34	70	153
		4	52	13	1	13	52	70	201
	8	1	16	5	1	5	16	50	93
		2	38	8	1	8	38	50	143
		3	34	7	1	7	34	70	153
		4	52	13	1	13	52	70	201



LOC	TAR	MOD	APP	ING	ATT	EGR	RET	TUR	SUM
5	9	1	17	11	1	11	17	50	107
		2	33	15	1	15	33	50	147
	10	1	17	11	1	11	17	50	107
		2	33	15	1	15	33	50	147
		3	30	14	1	14	30	70	159
		4	42	23	1	23	42	70	201
6	11	1	33	33	1	33	33	70	203
		2	40	48	1	48	40	70	247
	12	1	17	26	1	26	17	50	137
		2	37	37	1	37	37	50	199
		3	33	33	1	33	33	70	203
		4	40	48	1	48	40	70	247
	13	1	17	26	1	26	17	50	137
		2	37	37	1	37	37	50	199
7	14	1	17	17	1	17	17	50	119
		2	39	26	1	26	39	50	181
	15	1	17	17	1	17	17	50	119
		2	39	26	1	26	39	50	181
		3	35	24	1	24	35	70	189
		4	46	37	1	37	46	70	237
	16	1	17	17	1	17	17	50	119
		2	39	26	1	26	39	50	181
		3	35	24	1	24	35	70	189
		4	46	37	1	37	46	70	237

LOC	TAR	MOD	APP	ING	ATT	EGR	RET	TUR	SUM
8	17	1	29	34	1	34	29	70	197
		2	35	49	1	49	35	70	239
	18	1	29	34	1	34	29	70	197
		2	35	49	1	49	35	70	239
	19	1	16	28	1	28	16	50	139
		2	33	38	1	38	33	50	193
		3	29	34	1	34	29	70	197
		4	35	49	1	49	35	70	239
9	20	1	16	22	1	22	16	50	127
		2	34	31	1	31	34	50	181
	21	1	30	28	1	28	30	70	187
		2	38	41	1	41	38	70	229
	22	1	16	22	1	22	16	50	127
		2	34	31	1	31	34	50	181
10	23	1	16	18	1	18	16	50	119
		2	32	24	1	24	32	50	163
		3	29	22	1	22	29	70	173
		4	38	34	1	34	38	70	215
	24	1	16	18	1	18	16	50	119
		2	32	24	1	24	32	50	163
	25	1	16	18	1	18	16	50	119
		2	32	24	1	24	32	50	163
		3	29	22	1	22	29	70	173
		4	38	34	1	34	38	70	215

LOC	TAR	MOD	APP	ING	ATT	EGR	RET	TUR	SUM	
11	26	1	39	10	1	10	39	50	149	
		2	24	4	1	4	24	50	107	
		3	22	4	1	4	22	70	123	
		4	28	6	1	6	28	70	139	
	27	1	22	4	1	4	22	70	123	
		2	28	6	1	6	28	70	139	
	28	1	34	19	1	19	34	50	157	
		2	25	9	1	9	25	50	119	
12		3	22	8	1	8	22	70	131	
		4	24	11	1	11	24	70	141	
29	1	22	8	1	8	22	70	131		
	2	24	11	1	11	24	70	141		
30	1	24	6	1	6	24	70	131		
	2	22	7	1	7	22	70	129		
31	1	42	16	1	16	42	50	167		
	2	26	7	1	7	26	50	117		
	13		3	24	6	1	6	24	70	131
			4	22	7	1	7	22	70	129
32	1	40	28	1	28	40	50	187		
	2	28	13	1	13	28	50	133		
	3	26	12	1	12	26	70	147		
	4	19	10	1	10	19	70	129		
33	1	26	12	1	12	26	70	147		
	2	19	10	1	10	19	70	129		
14	32	1	40	28	1	28	40	50	187	
		2	28	13	1	13	28	50	133	
		3	26	12	1	12	26	70	147	
		4	19	10	1	10	19	70	129	
	33	1	26	12	1	12	26	70	147	
		2	19	10	1	10	19	70	129	

LOC	TAR	MOD	APP	ING	ATT	EGR	RET	TUR	SUM
15	34	1	30	10	1	10	30	70	151
		2	18	7	1	7	18	70	121
	35	1	50	25	1	25	50	50	201
		2	33	11	1	11	33	50	139
		3	30	10	1	10	30	70	151
		4	18	7	1	7	18	70	121
16	36	1	25	36	1	36	25	50	173
		2	24	23	1	23	24	50	145
		3	22	21	1	21	22	70	157
		4	21	24	1	24	21	70	161
	37	1	22	21	1	21	22	70	157
		2	21	24	1	24	21	70	161
	38	1	25	36	1	36	25	50	173
		2	24	23	1	23	24	50	145
	39	1	22	16	1	16	22	70	147
		2	21	18	1	18	21	70	149
		1	29	31	1	31	29	50	171
		2	25	18	1	18	25	50	137
17	40	1	29	31	1	31	29	50	171
		2	25	18	1	18	25	50	137
	41	1	29	31	1	31	29	50	171
		2	25	18	1	18	25	50	137
	42	1	23	27	1	27	23	70	171
		2	19	26	1	26	19	70	161
18	43	1	23	27	1	27	23	70	171
		2	19	26	1	26	19	70	161
	44	1	23	27	1	27	23	70	171
		2	19	26	1	26	19	70	161

LOC	TAR	MOD	APP	ING	ATT	EGR	RET	TUR	SUM
19	45	1	32	38	1	38	32	50	191
		2	27	21	1	21	27	50	147
	46	1	24	19	1	19	24	70	157
		2	19	18	1	18	19	70	145
	47	1	32	38	1	38	32	50	191
		2	27	21	1	21	27	50	147
		3	24	19	1	19	24	70	157
		4	19	18	1	18	19	70	145
20	48	1	26	21	1	21	26	70	165
		2	18	18	1	18	18	70	143
	49	1	34	43	1	43	34	50	205
		2	28	24	1	24	28	50	155
		3	26	21	1	21	26	70	165
		4	18	18	1	18	18	70	143
	50	1	34	43	1	43	34	50	205
		2	28	24	1	24	28	50	155
		3	26	21	1	21	26	70	165
		4	18	18	1	18	18	70	143
21	51	1	17	25	1	25	17	50	135
		2	28	28	1	28	28	50	163
		3	25	25	1	25	25	70	171
		4	30	37	1	37	30	70	205
	52	1	17	25	1	25	17	50	135
		2	28	28	1	28	28	50	163
		3	25	25	1	25	25	70	171
		4	30	37	1	37	30	70	205

LOC	TAR	MOD	APP	ING	ATT	EGR	RET	TUR	SUM	
22	53	1	18	18	1	18	18	50	123	
		2	27	18	1	18	27	50	141	
	54	1	18	18	1	18	18	50	123	
		2	27	18	1	18	27	50	141	
		3	25	16	1	16	25	70	153	
		4	32	26	1	26	32	70	187	
23	55	1	23	24	1	24	23	70	165	
		2	27	34	1	34	27	70	193	
	56	1	18	28	1	28	18	50	143	
		2	26	27	1	27	26	50	157	
		3	23	24	1	24	23	70	165	
		4	27	34	1	34	27	70	193	
24	57	1	20	28	1	28	20	50	147	
		2	25	23	1	23	25	50	147	
		3	22	20	1	20	22	70	155	
		4	25	28	1	28	25	70	177	
	58	1	20	28	1	28	20	50	147	
		2	25	23	1	23	25	50	147	
	59	1	21	36	1	36	21	50	165	
		2	24	28	1	28	24	50	155	
25		3	22	25	1	25	22	70	165	
		4	23	32	1	32	23	70	181	
60	1	22	25	1	25	22	70	165		
	2	23	32	1	32	23	70	181		

LOC	TAR	MOD	APP	ING	ATT	EGR	RET	TUR	SUM
26	61	1	17	35	1	35	17	50	155
		2	26	37	1	37	26	50	177
	62	1	17	35	1	35	17	50	155
		2	26	37	1	37	26	50	177
		3	24	34	1	34	24	70	187
		4	26	45	1	45	26	70	213
	63	1	24	34	1	34	24	70	187
		2	26	45	1	45	26	70	213
27	64	1	17	41	1	41	17	50	167
		2	26	42	1	42	26	50	187
	65	1	17	41	1	41	17	50	167
		2	26	42	1	42	26	50	187
		3	23	38	1	38	23	70	193
		4	25	48	1	48	25	70	217
	66	1	17	41	1	41	17	50	167
		2	26	42	1	42	26	50	187
28	67	1	23	30	1	30	23	70	177
		2	25	40	1	40	25	70	201
	68	1	17	35	1	35	17	50	155
		2	25	34	1	34	25	50	169
	69	1	17	35	1	35	17	50	155
		2	25	34	1	34	25	50	169
		3	23	30	1	30	23	70	177
		4	25	40	1	40	25	70	201

LOC	TAR	MOD	APP	ING	ATT	EGR	RET	TUR	SUM
29	70	1	18	40	1	40	18	50	167
		2	25	37	1	37	25	50	175
		3	22	33	1	33	22	70	181
		4	23	42	1	42	23	70	201
	71	1	18	40	1	40	18	50	167
		2	25	37	1	37	25	50	175
		3	22	33	1	33	22	70	181
		4	23	42	1	42	23	70	201
	72	1	18	40	1	40	18	50	167
		2	25	37	1	37	25	50	175
		3	22	33	1	33	22	70	181
		4	23	42	1	42	23	70	201
30	73	1	19	37	1	37	19	50	163
		2	24	32	1	32	24	50	163
		3	22	28	1	28	22	70	171
		4	23	36	1	36	23	70	189
	74	1	19	37	1	37	19	50	163
		2	24	32	1	32	24	50	163
		3	22	28	1	28	22	70	171
		4	23	36	1	36	23	70	189
	75	1	19	37	1	37	19	50	163
		2	24	32	1	32	24	50	163
		3	22	28	1	28	22	70	171
		4	23	36	1	36	23	70	189



LOC	TAR	MOD	APP	ING	ATT	EGR	RET	TUR	SUM
31	76	1	28	23	1	23	28	70	173
		2	18	18	1	18	18	70	143
	77	1	37	47	1	47	37	50	219
		2	31	26	1	26	31	50	165
		3	28	23	1	23	28	70	173
		4	18	18	1	18	18	70	143
32	78	1	37	52	1	52	37	50	229
		2	32	30	1	30	32	50	175
		3	28	27	1	27	28	70	181
		4	18	21	1	21	18	70	149
	79	1	37	52	1	52	37	50	229
		2	32	30	1	30	32	50	175
		3	28	27	1	27	28	70	181
		4	18	21	1	21	18	70	149
33	80	1	45	45	1	45	45	50	231
		2	36	24	1	24	36	50	171
		3	33	22	1	22	33	70	181
		4	19	15	1	15	19	70	139
	81	1	45	45	1	45	45	50	231
		2	36	24	1	24	36	50	171
34	82	1	34	25	1	25	34	70	189
		2	20	18	1	18	20	70	147
	83	1	45	51	1	51	45	50	243
		2	37	28	1	28	37	50	181

LOC	TAR	MOD	APP	ING	ATT	EGR	RET	TUR	SUM
35	84	1	44	61	1	61	44	50	261
		2	39	35	1	35	39	50	199
		3	35	32	1	32	35	70	205
		4	21	23	1	23	21	70	159
	85	1	35	32	1	32	35	70	205
		2	21	23	1	23	21	70	159
	86	1	27	33	1	33	27	70	191
		2	18	27	1	27	18	70	161
36	87	1	32	60	1	60	32	50	235
		2	30	37	1	37	30	50	185
		3	27	33	1	33	27	70	191
		4	18	27	1	27	18	70	161
	88	1	32	60	1	60	32	50	235
		2	30	37	1	37	30	50	185
		3	27	33	1	33	27	70	191
		4	18	27	1	27	18	70	161
	89	1	27	40	1	40	27	70	205
		2	18	33	1	33	18	70	173
	90	1	27	40	1	40	27	70	205
		2	18	33	1	33	18	70	173
	91	1	27	40	1	40	27	70	205
		2	18	33	1	33	18	70	173

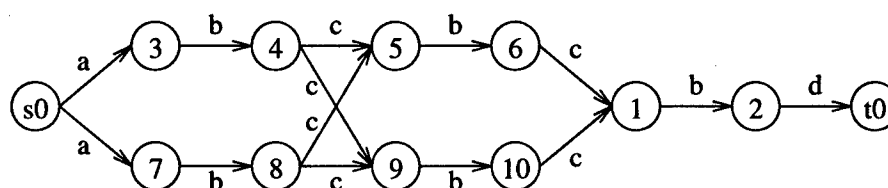
LOC	TAR	MOD	APP	ING	ATT	EGR	RET	TUR	SUM
38	92	1	38	61	1	61	38	50	249
		2	34	37	1	37	34	50	193
	93	1	38	61	1	61	38	50	249
		2	34	37	1	37	34	50	193
		3	31	33	1	33	31	70	199
		4	19	25	1	25	19	70	159
	94	1	38	61	1	61	38	50	249
		2	34	37	1	37	34	50	193
39	95	1	35	70	1	70	35	50	261
		2	33	44	1	44	33	50	205
		3	30	40	1	40	30	70	211
		4	19	31	1	31	19	70	171
	96	1	35	70	1	70	35	50	261
		2	33	44	1	44	33	50	205
		3	30	40	1	40	30	70	211
		4	19	31	1	31	19	70	171
	97	1	35	70	1	70	35	50	261
		2	33	44	1	44	33	50	205
		3	30	40	1	40	30	70	211
		4	19	31	1	31	19	70	171

LOC	TAR	MOD	APP	ING	ATT	EGR	RET	TUR	SUM
40	98	1	34	77	1	77	34	50	273
		2	33	50	1	50	33	50	217
		3	30	45	1	45	30	70	221
		4	20	36	1	36	20	70	183
	99	1	34	77	1	77	34	50	273
		2	33	50	1	50	33	50	217
		3	30	45	1	45	30	70	221
		4	20	36	1	36	20	70	183
	100	1	30	45	1	45	30	70	221
		2	20	36	1	36	20	70	183

#### B.4 Precedence Networks

This section contains the precedence networks for every subproblem in the case study sample problem. The problem is decomposed first by attack wave and then by mission type. Each subproblem  $i$  has a dummy source node,  $s_i$ , and a dummy sink node  $t_i$ . The labels on the other nodes in the precedence networks correspond to the identification numbers of the targets in the sample problem. The letters labeling the arcs in the precedence networks indicate the type of generalized precedence enforced between the targets that the arcs connect. The variations of generalized precedence used are explained in Chapter VII.

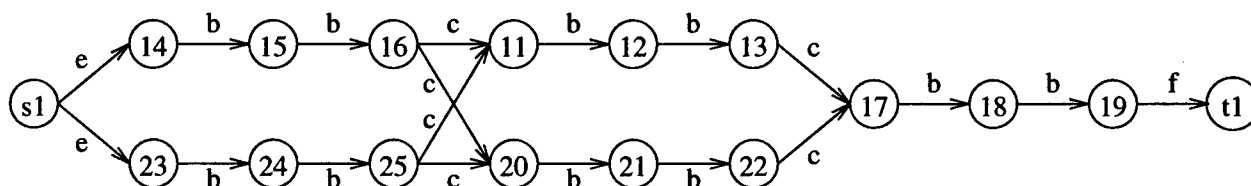
This is the precedence network for the first wave of missions.



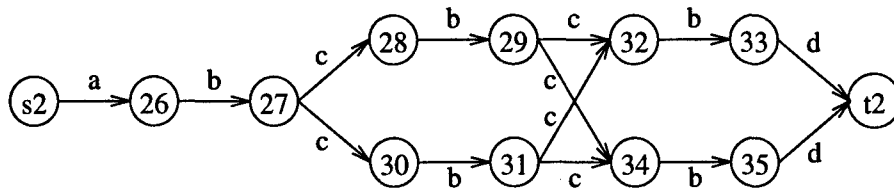
#### Enemy Air Defense Targets

---

#### Interdiction Targets

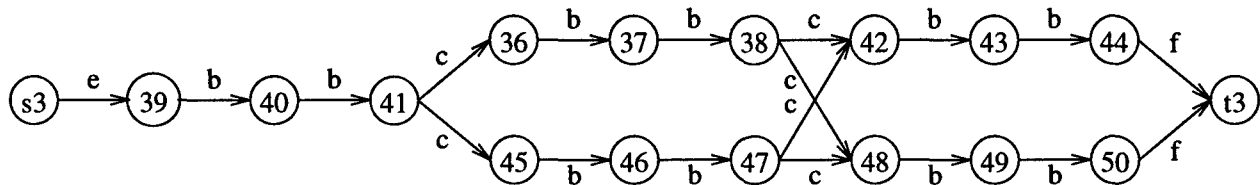


This is the precedence network for the second wave of missions.

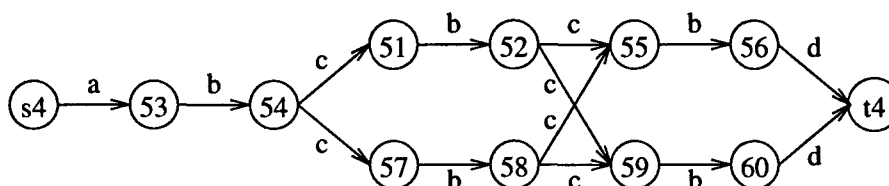


### Enemy Air Defense Targets

### Interdiction Targets

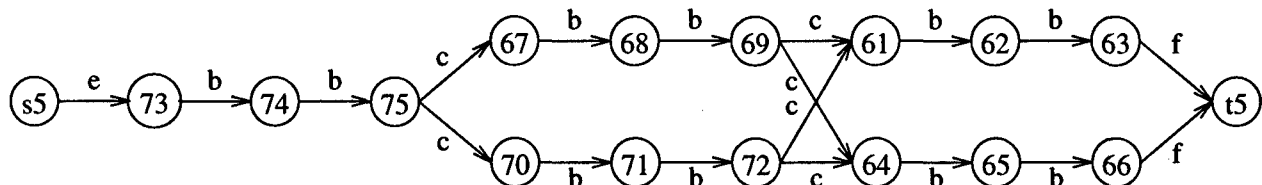


This is the precedence network for the third wave of missions.

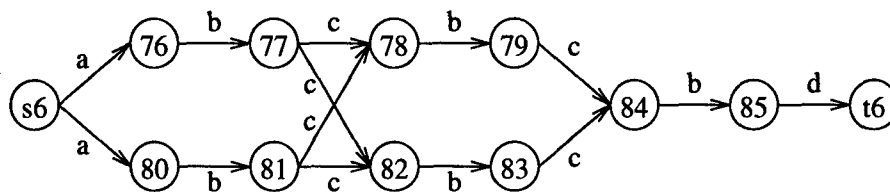


### Enemy Air Defense Targets

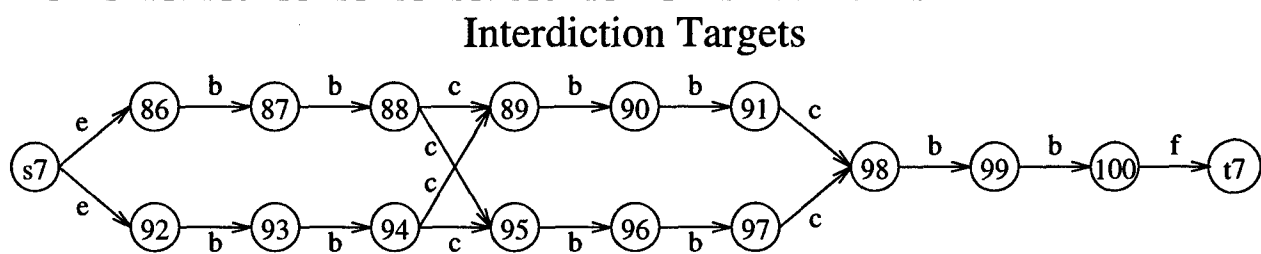
### Interdiction Targets



This is the precedence network for the fourth wave of missions.



### Enemy Air Defense Targets



### Interdiction Targets

### B.5 Lag Values for Generalized Precedence

This section contains the minimum lag values required to formulate the generalized precedence constraints for the case study sample problem. Each table in this section contains all of the possible values for the minimum lag enforced between the start times of an activity and each of its successor activities. The lag value enforced depends on both the execution mode of the predecessor activity,  $m$ , and the execution mode of the successor activity,  $n$ .

Predecessor	Successor target 3				Successor target 7			
	$n = 1$	$n = 2$	$n = 3$	$n = 4$	$n = 1$	$n = 2$	$n = 3$	$n = 4$
$source_0 \quad m = 1$	0	0	0	0	0	0	0	0

Predecessor		Successor target 2	
		$n = 1$	$n = 2$
target 1	$m = 1$	1	-24
	$m = 2$	26	1

Predecessor		Successor $sink_0$
		$n = 1$
target 2	$m = 1$	64
	$m = 2$	89

Predecessor		Successor target 4	
		$n = 1$	$n = 2$
target 3	$m = 1$	-29	-54
	$m = 2$	6	-19
	$m = 3$	1	-24
	$m = 4$	26	1



Predecessor		Successor target 5				Successor target 9	
		$n = 1$	$n = 2$	$n = 3$	$n = 4$	$n = 1$	$n = 2$
target 4	$m = 1$	36	5	11	-13	32	12
	$m = 2$	61	30	36	12	57	37

Predecessor		Successor target 6	
		$n = 1$	$n = 2$
target 5	$m = 1$	1	-30
	$m = 2$	32	1
	$m = 3$	26	-5
	$m = 4$	50	19

Predecessor		Successor target 1	
		$n = 1$	$n = 2$
target 6	$m = 1$	-33	-58
	$m = 2$	-2	-27

Predecessor		Successor target 8			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 7	$m = 1$	1	-24	-19	-43
	$m = 2$	26	1	6	-18
	$m = 3$	21	-4	1	-23
	$m = 4$	45	20	25	1

Predecessor		Successor target 5				Successor target 9	
		$n = 1$	$n = 2$	$n = 3$	$n = 4$	$n = 1$	$n = 2$
target 8	$m = 1$	3	-28	-22	-46	-1	-21
	$m = 2$	28	-3	3	-21	24	4
	$m = 3$	23	-8	-2	-26	19	-1
	$m = 4$	47	16	22	-2	43	23

Predecessor		Successor target 10			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 9	$m = 1$	1	-19	-15	-36
	$m = 2$	21	1	5	-16

Predecessor		Successor target 1	
		$n = 1$	$n = 2$
target 10	$m = 1$	-29	-54
	$m = 2$	-9	-34
	$m = 3$	-13	-38
	$m = 4$	8	-17

Predecessor		Successor target 14		Successor target 23			
		$n = 1$	$n = 2$	$n = 1$	$n = 2$	$n = 3$	$n = 4$
<i>source</i> <sub>1</sub>	$m = 1$	22	0	23	7	10	1

Predecessor		Successor target 12			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 11	$m = 1$	24	-7	1	-21
	$m = 2$	46	15	23	1

Predecessor		Successor	
		target 13	
		$n = 1$	$n = 2$
target 12	$m = 1$	1	-30
	$m = 2$	32	1
	$m = 3$	24	-7
	$m = 4$	46	15

Predecessor		Successor	
		target 17	
		$n = 1$	$n = 2$
target 13	$m = 1$	-14	-35
	$m = 2$	17	-4

Predecessor		Successor			
		target 15			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 14	$m = 1$	1	-30	-24	-48
	$m = 2$	32	1	7	-17

Predecessor		Successor			
		target 16			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 15	$m = 1$	1	-30	-24	-48
	$m = 2$	32	1	7	-17
	$m = 3$	26	-5	1	-23
	$m = 4$	50	19	25	1

Predecessor		Successor		Successor	
		target 11		target 20	
		$n = 1$	$n = 2$	$n = 1$	$n = 2$
target 16	$m = 1$	-26	-48	2	-25
	$m = 2$	5	-17	33	6
	$m = 3$	-1	-23	27	0
	$m = 4$	23	1	51	24

Predecessor		Successor	
		target 18	
		$n = 1$	$n = 2$
target 17	$m = 1$	1	-20
	$m = 2$	22	1

Predecessor		Successor			
		target 19			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 18	$m = 1$	20	-7	1	-20
	$m = 2$	41	14	22	1

Predecessor		Successor
		$sink_1$
		$n = 1$
target 19	$m = 1$	139
	$m = 2$	193
	$m = 3$	197
	$m = 4$	239

Predecessor	Successor target 21 $n = 1$ $n = 2$
target 20 $m = 1$	-19 -40
$m = 2$	8 -13

Predecessor	Successor target 22 $n = 1$ $n = 2$
target 21 $m = 1$	21 -6
$m = 2$	42 15

Predecessor	Successor target 17 $n = 1$ $n = 2$
target 22 $m = 1$	-19 -40
$m = 2$	8 -13

Predecessor	Successor target 24 $n = 1$ $n = 2$
target 23 $m = 1$	1 -21
$m = 2$	23 1
$m = 3$	18 -4
$m = 4$	39 17

Predecessor	Successor target 25 $n = 1$ $n = 2$ $n = 3$ $n = 4$
target 24 $m = 1$	1 -21 -16 -37
$m = 2$	23 1 6 -15

Predecessor		Successor		Successor	
		target 11		target 20	
		$n = 1$	$n = 2$	$n = 1$	$n = 2$
target 25	$m = 1$	-26	-48	2	-25
	$m = 2$	-4	-26	24	-3
	$m = 3$	-9	-31	19	-8
	$m = 4$	12	-10	40	13

Predecessor		Successor			
		target 26			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
$source_2$	$m = 1$	0	0	0	0

Predecessor		Successor	
		target 27	
		$n = 1$	$n = 2$
target 26	$m = 1$	3	-5
	$m = 2$	1	-7
	$m = 3$	9	1
	$m = 4$	24	16

Predecessor		Successor				Successor	
		target 28				target 30	
		$n = 1$	$n = 2$	$n = 3$	$n = 4$	$n = 1$	$n = 2$
target 27	$m = 1$	-2	2	-3	-21	3	2
	$m = 2$	6	10	5	-13	11	10

Predecessor		Successor target 29	
		$n = 1$	$n = 2$
target 28	$m = 1$	5	0
	$m = 2$	1	-4
	$m = 3$	6	1
	$m = 4$	24	19

Predecessor		Successor target 32				Successor target 34	
		$n = 1$	$n = 2$	$n = 3$	$n = 4$	$n = 1$	$n = 2$
target 29	$m = 1$	7	-5	-2	-32	11	-4
	$m = 2$	12	0	3	-27	16	1

Predecessor		Successor target 31			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 30	$m = 1$	-3	1	0	-28
	$m = 2$	-2	2	1	-27

Predecessor		Successor target 32				Successor target 34	
		$n = 1$	$n = 2$	$n = 3$	$n = 4$	$n = 1$	$n = 2$
target 31	$m = 1$	10	-2	1	-29	14	-1
	$m = 2$	6	-6	-3	-33	10	-5
	$m = 3$	7	-5	-2	-32	11	-4
	$m = 4$	35	23	26	-4	39	24

Predecessor		Successor	
		target 33	
		$n = 1$	$n = 2$
target 32	$m = 1$	1	-8
	$m = 2$	13	4
	$m = 3$	10	1
	$m = 4$	40	31

Predecessor	Successor
	$sink_2$
	$n = 1$
target 33 $m = 1$	30
$m = 2$	39

Predecessor		Successor			
		target 35			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 34	$m = 1$	1	-18	-14	-49
	$m = 2$	16	-3	1	-34

Predecessor	Successor
	$sink_2$
	$n = 1$
target 35 $m = 1$	26
$m = 2$	45
$m = 3$	41
$m = 4$	76

Predecessor	Successor
	target 39
	$n = 1$ $n = 2$
$source_3$ $m = 1$	0   1



Predecessor		Successor target 37 $n = 1$ $n = 2$	
target 36	$m = 1$	5	3
	$m = 2$	1	-1
	$m = 3$	3	1
	$m = 4$	19	17

Predecessor		Successor target 38 $n = 1$ $n = 2$	
target 37	$m = 1$	-3	-17
	$m = 2$	-1	-15

Predecessor		Successor target 42 $n = 1$ $n = 2$		Successor target 48 $n = 1$ $n = 2$	
target 38	$m = 1$	8	3	17	6
	$m = 2$	22	17	31	20

Predecessor		Successor target 40 $n = 1$ $n = 2$	
target 39	$m = 1$	-4	-21
	$m = 2$	-3	-20

Predecessor		Successor target 41 $n = 1$ $n = 2$	
target 40	$m = 1$	1	-16
	$m = 2$	18	1

Predecessor		Successor target 36				Successor target 45	
		$n = 1$	$n = 2$	$n = 3$	$n = 4$	$n = 1$	$n = 2$
target 41	$m = 1$	2	6	4	-12	1	-21
	$m = 2$	19	23	21	5	18	-4

Predecessor		Successor target 43	
		$n = 1$	$n = 2$
target 42	$m = 1$	1	-4
	$m = 2$	6	1

Predecessor		Successor target 44	
		$n = 1$	$n = 2$
target 43	$m = 1$	1	-4
	$m = 2$	6	1

Predecessor		Successor $sink_3$
		$n = 1$
target 44	$m = 1$	161
	$m = 2$	171

Predecessor		Successor target 46	
		$n = 1$	$n = 2$
target 45	$m = 1$	12	6
	$m = 2$	34	28

Predecessor		Successor			
		target 47			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 46	$m = 1$	1	-10	-5	-32
	$m = 2$	7	-4	1	-26

Predecessor		Successor		Successor	
		target 42		target 48	
		$n = 1$	$n = 2$	$n = 1$	$n = 2$
target 47	$m = 1$	-2	-7	7	-4
	$m = 2$	9	4	18	7
	$m = 3$	4	-1	13	2
	$m = 4$	31	26	40	29

Predecessor		Successor			
		target 49			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 48	$m = 1$	1	-15	-10	-40
	$m = 2$	12	-4	1	-29

Predecessor		Successor			
		target 50			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 49	$m = 1$	1	-15	-10	-40
	$m = 2$	17	1	6	-24
	$m = 3$	12	-4	1	-29
	$m = 4$	42	26	31	1

Predecessor	Successor
	$sink_3$ $n = 1$
target 50 $m = 1$	143
$m = 2$	155
$m = 3$	165
$m = 4$	205

Predecessor	Successor
	target 53 $n = 1$ $n = 2$
$source_4$ $m = 1$	0            0

Predecessor	Successor
	target 52 $n = 1$ $n = 2$ $n = 3$ $n = 4$
target 51 $m = 1$	1        -13        -7        -24
$m = 2$	15        1            7        -10
$m = 3$	9          -5           1        -16
$m = 4$	26        12          18        1

Predecessor	Successor	Successor
	target 55	target 59
	$n = 1$ $n = 2$	$n = 1$ $n = 2$ $n = 3$ $n = 4$
target 52 $m = 1$	1        -13	-4        -9        1        -7
$m = 2$	15        1	10        5        15        7
$m = 3$	9          -5	4          -1        9        1
$m = 4$	26        12	21        16        26        18

Predecessor		Successor target 54			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 53	$m = 1$	1	-8	-4	-21
	$m = 2$	10	1	5	-12

Predecessor		Successor target 51				Successor target 57			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$	$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 54	$m = 1$	0	-14	-8	-25	-6	-6	0	-11
	$m = 2$	9	-5	1	-16	3	3	9	-2
	$m = 3$	5	-9	-3	-20	-1	-1	5	-6
	$m = 4$	22	8	14	-3	16	16	22	11

Predecessor		Successor target 56			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 55	$m = 1$	2	-5	1	-13
	$m = 2$	16	9	15	1

Predecessor		Successor $sink_4$ $n = 1$
target 56	$m = 1$	47
	$m = 2$	54
	$m = 3$	48
	$m = 4$	62

Predecessor	Successor target 58 $n = 1$ $n = 2$	
target 57 $m = 1$	1	1
$m = 2$	1	1
$m = 3$	-5	-5
$m = 4$	6	6

Predecessor	Successor target 55 $n = 1$ $n = 2$		Successor target 59 $n = 1$ $n = 2$ $n = 3$ $n = 4$			
target 58 $m = 1$	7	-7	2	-3	7	-1
$m = 2$	7	-7	2	-3	7	-1

Predecessor	Successor target 60 $n = 1$ $n = 2$	
target 59 $m = 1$	6	-2
$m = 2$	11	3
$m = 3$	1	-7
$m = 4$	9	1

Predecessor	Successor $sink_4$ $n = 1$
target 60 $m = 1$	48
$m = 2$	56

Predecessor	Successor target 73 $n = 1$ $n = 2$ $n = 3$ $n = 4$			
$source_5$ $m = 1$	5	0	2	1

Predecessor		Successor			
		target 62			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 61	$m = 1$	1	-10	-5	-18
	$m = 2$	12	1	6	-7

Predecessor		Successor	
		target 63	
		$n = 1$	$n = 2$
target 62	$m = 1$	-5	-18
	$m = 2$	6	-7
	$m = 3$	1	-12
	$m = 4$	14	1

Predecessor		Successor
		$sink_5$
		$n = 1$
target 63	$m = 1$	187
	$m = 2$	213

Predecessor		Successor			
		target 65			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 64	$m = 1$	1	-9	-2	-14
	$m = 2$	11	1	8	-4

Predecessor		Successor	
		target 66	
		$n = 1$	$n = 2$
target 65	$m = 1$	1	-9
	$m = 2$	11	1
	$m = 3$	4	-6
	$m = 4$	16	6

Predecessor		Successor
		$sink_5$
		$n = 1$
target 66	$m = 1$	167
	$m = 2$	187

Predecessor		Successor	
		target 68	
		$n = 1$	$n = 2$
target 67	$m = 1$	2	-5
	$m = 2$	14	7

Predecessor		Successor			
		target 69			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 68	$m = 1$	1	-6	0	-12
	$m = 2$	8	1	7	-5



Predecessor		Successor target 61		Successor target 64	
		$n = 1$	$n = 2$	$n = 1$	$n = 2$
target 69	$m = 1$	6	-5	0	-10
	$m = 2$	13	2	7	-3
	$m = 3$	7	-4	1	-9
	$m = 4$	19	8	13	3

Predecessor		Successor target 71			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 70	$m = 1$	1	-3	4	-6
	$m = 2$	5	1	8	-2
	$m = 3$	-2	-6	1	-9
	$m = 4$	8	4	11	1

Predecessor		Successor target 72			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 71	$m = 1$	1	-3	4	-6
	$m = 2$	5	1	8	-2
	$m = 3$	-2	-6	1	-9
	$m = 4$	8	4	11	1

Predecessor		Successor target 61		Successor target 64	
		$n = 1$	$n = 2$	$n = 1$	$n = 2$
target 72	$m = 1$	12	1	6	-4
	$m = 2$	16	5	10	0
	$m = 3$	9	-2	3	-7
	$m = 4$	19	8	13	3

Predecessor		Successor			
		target 74			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 73	$m = 1$	1	1	7	-2
	$m = 2$	1	1	7	-2
	$m = 3$	-5	-5	1	-8
	$m = 4$	4	4	10	1

Predecessor		Successor			
		target 75			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 74	$m = 1$	1	1	7	-2
	$m = 2$	1	1	7	-2
	$m = 3$	-5	-5	1	-8
	$m = 4$	4	4	10	1

Predecessor		Successor		Successor			
		target 67		target 70			
		$n = 1$	$n = 2$	$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 75	$m = 1$	9	-3	4	0	7	-3
	$m = 2$	9	-3	4	0	7	-3
	$m = 3$	3	-9	-2	-6	1	-9
	$m = 4$	12	0	7	3	10	0

Predecessor		Successor		Successor			
		target 76		target 80			
		$n = 1$	$n = 2$	$n = 1$	$n = 2$	$n = 3$	$n = 4$
<i>source</i> <sub>6</sub>	$m = 1$	0	0	0	0	0	0

Predecessor		Successor target 77			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 76	$m = 1$	1	-20	-14	-47
	$m = 2$	16	-5	1	-32

Predecessor		Successor target 78				Successor target 82	
		$n = 1$	$n = 2$	$n = 3$	$n = 4$	$n = 1$	$n = 2$
target 77	$m = 1$	3	-20	-13	-47	4	-17
	$m = 2$	24	1	8	-26	25	4
	$m = 3$	18	-5	2	-32	19	-2
	$m = 4$	51	28	35	1	52	31

Predecessor		Successor target 79			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 78	$m = 1$	1	-22	-15	-49
	$m = 2$	24	1	8	-26
	$m = 3$	17	-6	1	-33
	$m = 4$	51	28	35	1

Predecessor		Successor target 84			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 79	$m = 1$	1	-29	-22	-60
	$m = 2$	24	-6	1	-37
	$m = 3$	17	-13	-6	-44
	$m = 4$	51	21	28	-10

Predecessor		Successor target 81	
		$n = 1$	$n = 2$
target 80	$m = 1$	-25	-55
	$m = 2$	1	-29
	$m = 3$	-4	-34
	$m = 4$	31	1

Predecessor		Successor target 78				Successor target 82	
		$n = 1$	$n = 2$	$n = 3$	$n = 4$	$n = 1$	$n = 2$
target 81	$m = 1$	27	4	11	-23	28	7
	$m = 2$	57	34	41	7	58	37

Predecessor		Successor target 83	
		$n = 1$	$n = 2$
target 82	$m = 1$	-26	-57
	$m = 2$	-5	-36

Predecessor		Successor target 84			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 83	$m = 1$	27	-3	4	-34
	$m = 2$	58	28	35	-3

Predecessor	Successor	
	target 85	
	$n = 1$	$n = 2$
target 84	$m = 1$	1      -22
	$m = 2$	31      8
	$m = 3$	24      1
	$m = 4$	62      39

Predecessor	Successor
	$sink_6$
	$n = 1$
target 85 $m = 1$	45
$m = 2$	68

Predecessor	Successor		Successor	
	target 86		target 92	
	$n = 1$	$n = 2$	$n = 1$	$n = 2$
$source_7$ $m = 1$	20	11	4	0

Predecessor		Successor			
		target 87			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 86	$m = 1$	1	-21	-14	-46
	$m = 2$	16	-6	1	-31

Predecessor		Successor			
		target 88			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 87	$m = 1$	1	-21	-14	-46
	$m = 2$	23	1	8	-24
	$m = 3$	16	-6	1	-31
	$m = 4$	48	26	33	1

Predecessor		Successor		Successor			
		target 89		target 95			
		$n = 1$	$n = 2$	$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 88	$m = 1$	0	-16	1	-26	-19	-54
	$m = 2$	22	6	23	-4	3	-32
	$m = 3$	15	-1	16	-11	-4	-39
	$m = 4$	47	31	48	21	28	-7

Predecessor		Successor	
		target 90	
		$n = 1$	$n = 2$
target 89	$m = 1$	1	-15
	$m = 2$	17	1

Predecessor		Successor	
		target 91	
		$n = 1$	$n = 2$
target 90	$m = 1$	1	-15
	$m = 2$	17	1

Predecessor		Successor			
		target 98			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 91	$m = 1$	1	-26	-18	-54
	$m = 2$	17	-10	-2	-38

Predecessor		Successor			
		target 93			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 92	$m = 1$	28	1	8	-27
	$m = 2$	56	29	36	1

Predecessor		Successor target 94	
		$n = 1$	$n = 2$
target 93	$m = 1$	-26	-54
	$m = 2$	1	-27
	$m = 3$	-6	-34
	$m = 4$	29	1

Predecessor		Successor target 89		Successor target 95			
		$n = 1$	$n = 2$	$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 94	$m = 1$	26	10	27	0	7	-28
	$m = 2$	54	38	55	28	35	0

Predecessor		Successor target 96			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 95	$m = 1$	1	-26	-19	-54
	$m = 2$	28	1	8	-27
	$m = 3$	21	-6	1	-34
	$m = 4$	56	29	36	1

Predecessor		Successor target 97			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 96	$m = 1$	1	-26	-19	-54
	$m = 2$	28	1	8	-27
	$m = 3$	21	-6	1	-34
	$m = 4$	56	29	36	1

Predecessor		Successor target 98			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 97	$m = 1$	0	-27	-19	-55
	$m = 2$	27	0	8	-28
	$m = 3$	20	-7	1	-35
	$m = 4$	55	28	36	0

Predecessor		Successor target 99			
		$n = 1$	$n = 2$	$n = 3$	$n = 4$
target 98	$m = 1$	1	-26	-18	-54
	$m = 2$	28	1	9	-27
	$m = 3$	20	-7	1	-35
	$m = 4$	56	29	37	1

Predecessor		Successor target 100	
		$n = 1$	$n = 2$
target 99	$m = 1$	1	-18
	$m = 2$	28	9
	$m = 3$	20	1
	$m = 4$	56	37

Predecessor		Successor <i>sink<sub>7</sub></i> $n = 1$
target 100	$m = 1$	183
	$m = 2$	221



### *B.6 The Optimal Solution*

This section contains the optimal solution to the case study sample problem. The table provides the assignment of resources to targets and the important mission scheduling information that would be provided in the mission lines of the ATO in an actual force level planning exercise. It was assumed that the attack waves are temporally discrete and arbitrary start times of 0000, 0600, 1200, and 1800 were assigned to the waves of the sample problem.

Target	Resources	Takeoff	Attack
1	4 AC-2 from Base B	0017	0120-0121
2	4 AC-2 from Base B	0018	0121-0122
3	4 AC-1 from Base A	0000	0024-0025
4	4 AC-2 from Base B	0000	0054-0055
5	2 AC-2 from Base C	0000	0113-0114
6	4 AC-1 from Base A	0050	0114-0115
7	2 AC-1 from Base A	0000	0021-0022
8	2 AC-2 from Base C	0000	0105-0106
9	4 AC-1 from Base A	0043	0111-0112
10	4 AC-1 from Base B	0024	0112-0113
11	2 AC-2 from Base B	0100	0206-0207
12	2 AC-1 from Base A	0124	0207-0208
13	2 AC-1 from Base A	0125	0208-0209
14	4 AC-1 from Base A	0123	0157-0158
15	2 AC-1 from Base A	0124	0158-0159
16	2 AC-2 from Base C	0036	0159-0200
17	4 AC-2 from Base C	0050	0214-0215
18	2 AC-2 from Base C	0051	0215-0216
19	2 AC-1 from Base A	0132	0216-0217
20	4 AC-1 from Base A	0128	0206-0207
21	2 AC-2 from Base C	0048	0207-0208
22	2 AC-1 from Base A	0130	0208-0209
23	2 AC-1 from Base A	0124	0158-0159
24	2 AC-1 from Base A	0125	0159-0200
25	2 AC-2 from Base C	0048	0200-0201

Target	Resources	Takeoff	Attack
26	4 AC-2 from Base B	0634	0700-0701
27	2 AC-2 from Base B	0635	0701-0702
28	2 AC-1 from Base B	0633	0707-0708
29	2 AC-2 from Base B	0638	0708-0709
30	2 AC-2 from Base C	0638	0707-0708
31	4 AC-1 from Base A	0610	0708-0709
32	2 AC-2 from Base C	0645	0714-0715
33	2 AC-2 from Base C	0646	0715-0716
34	2 AC-2 from Base C	0649	0714-0715
35	4 AC-1 from Base A	0600	0715-0716
36	4 AC-1 from Base B	0707	0754-0755
37	4 AC-2 from Base C	0710	0755-0756
38	2 AC-1 from Base B	0709	0756-0757
39	4 AC-2 from Base C	0707	0746-0747
40	4 AC-1 from Base A	0647	0747-0748
41	2 AC-1 from Base B	0705	0748-0749
42	2 AC-2 from Base C	0717	0802-0803
43	2 AC-2 from Base C	0718	0803-0804
44	2 AC-2 from Base C	0719	0804-0805
45	4 AC-1 from Base A	0644	0754-0755
46	4 AC-2 from Base B	0712	0755-0756
47	2 AC-1 from Base B	0708	0756-0757
48	4 AC-2 from Base C	0726	0802-0803
49	2 AC-1 from Base A	0646	0803-0804
50	2 AC-1 from Base B	0712	0804-0805

Target	Resources	Takeoff	Attack
51	2 AC-1 from Base A	1224	1306-1307
52	4 AC-2 from Base C	1200	1307-1308
53	2 AC-1 from Base A	1223	1259-1300
54	4 AC-1 from Base A	1224	1300-1301
55	2 AC-2 from Base C	1212	1313-1314
56	4 AC-2 from Base C	1213	1314-1315
57	2 AC-1 from Base B	1218	1306-1307
58	4 AC-1 from Base A	1219	1307-1308
59	2 AC-1 from Base A	1216	1313-1314
60	4 AC-2 from Base C	1219	1314-1315
61	4 AC-1 from Base B	1303	1406-1407
62	4 AC-2 from Base B	1309	1407-1408
63	2 AC-2 from Base B	1310	1408-1409
64	4 AC-1 from Base A	1308	1406-1407
65	4 AC-1 from Base B	1259	1407-1408
66	4 AC-1 from Base A	1310	1408-1409
67	4 AC-2 from Base C	1253	1358-1359
68	2 AC-1 from Base A	1307	1359-1400
69	2 AC-1 from Base B	1301	1400-1401
70	4 AC-1 from Base B	1256	1358-1359
71	4 AC-1 from Base A	1301	1359-1400
72	4 AC-1 from Base B	1258	1400-1401
73	4 AC-2 from Base B	1300	1350-1351
74	2 AC-1 from Base B	1255	1351-1352
75	2 AC-1 from Base A	1256	1352-1353

Target	Resources	Takeoff	Attack
76	2 AC-2 from Base C	1855	1931-1932
77	2 AC-1 from Base B	1835	1932-1933
78	2 AC-1 from Base B	1836	1938-1939
79	2 AC-2 from Base C	1900	1939-1940
80	2 AC-2 from Base C	1855	1929-1930
81	4 AC-1 from Base B	1830	1930-1931
82	2 AC-2 from Base B	1839	1938-1939
83	4 AC-1 from Base A	1803	1939-1940
84	4 AC-1 from Base A	1800	1945-1946
85	4 AC-2 from Base B	1839	1946-1947
86	4 AC-2 from Base B	1946	2046-2047
87	4 AC-1 from Base A	1915	2047-2048
88	2 AC-1 from Base A	1916	2048-2049
89	4 AC-2 from Base C	2007	2058-2059
90	2 AC-2 from Base C	2008	2059-2100
91	4 AC-2 from Base B	1953	2100-2101
92	2 AC-1 from Base B	1939	2050-2051
93	2 AC-2 from Base B	1947	2051-2052
94	2 AC-1 from Base B	1941	2052-2053
95	2 AC-1 from Base A	1913	2058-2059
96	2 AC-2 from Base C	2009	2059-2100
97	2 AC-1 from Base B	1943	2100-2101
98	4 AC-1 from Base B	1943	2106-2107
99	2 AC-2 from Base C	2011	2107-2108
100	4 AC-2 from Base C	2012	2108-2109

### Bibliography

1. Ahuja, Ravindra K., Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. New Jersey: Prentice-Hall, 1993.
2. Alcaraz, Javier and Concepción Maroto. "A Genetic Algorithm for the Resource-Constrained Project Scheduling Problem." *Sixth International Workshop on Project Management and Scheduling*. 7-10. Istanbul, Turkey: Boğaziçi University Printing Office, July 1998.
3. Babiak, Nicholas J. and Carol S. Lydic. "The Defense Mapping Agency and Tomorrow's Advanced Aerospace Warfare Systems." *IEEE 1990 National Aerospace and Electronics Conference*. 260-264. May 1990.
4. Balas, E. *Project Scheduling with Resource Constraints*, chapter in *Applications of Mathematical Programming Techniques* (E. M. L. Beale, editor), 187-200. American Elsevier Publishing Company, 1970.
5. Baumol, William J. and Tibor Fabian. "Decomposition, Pricing for Decentralization and External Economies," *Management Science*, 11:1-32 (1964).
6. Bazaraa, Mokhtar S., John J. Jarvis, and Hanif D. Sherali. *Linear Programming and Network Flows*. New York: Wiley, 1990.
7. Bector, Fayez F. "A New and Efficient Heuristic for Scheduling Projects With Resource Restrictions and Multiple Execution Modes," *European Journal of Operational Research*, 90(2):349-361 (April 1996).
8. Bowman, Edward H. "The Scheduling—Sequencing Problem," *Operations Research*, 7(5):621-624 (1959).
9. Brucker, Peter J. and Horst W. Hamacher. "K-Optimal Solution Sets for some Polynomially Solvable Scheduling Problems," *European Journal of Operational Research*, 41:194-202 (1989).
10. Carruthers, J. A. and A. Battersby. "Advances in Critical Path Methods," *Operations Research Quarterly*, 17:359-380 (1966).
11. Chamberlain, Derek B. and Charles R. Kirklen. "Advanced Mission Planning System (AMPS) Employing Terrain and Intelligence Database Support." *IEEE 1988 National Aerospace and Electronics Conference*. 1145-1151. May 1988.
12. Cheeseman, Peter, Bob Kanefsky, and William M. Taylor. "Where the Really Hard Problems Are," *Proceedings of IJCAI-91*, 331-337 (1991).
13. Chretienne, Philippe, Edward G. Coffman, Jr, Jan Karel Lenstra, and Zhen Lin, editors. *Scheduling Theory and its Applications*. New York: John Wiley and Sons, 1995.
14. Christofides, N., R. Alvares-Valdes, and J. M. Tamarit. "Project Scheduling with Resource Constraints: A Branch and Bound Approach," *European Journal of Operational Research*, 29(3):262-273 (June 1987).
15. Clausen, Jens and Jesper Larsson Träff. "Implementation of Parallel Branch-and-Bound Algorithms — Experiences with the Graph Partitioning Problem," *Annals of Operations Research*, 33:331-349 (1991).

16. Collins, Robert J. and David R. Jefferson. "Selection in Massively Parallel Genetic Algorithms." *Proceedings of the Fourth International Conference on Genetic Algorithms*, edited by Belew, R. K. and L. B. Booker. 244-248. Morgan Kaufmann, San Mateo California, 1991.
17. Dantzig, George B. and Philip Wolfe. "Decomposition Principle for Linear Programs," *Operations Research*, 8:101-111 (1960).
18. Davis, E. W. and G. E. Heidorn. "Optimal Project Scheduling under Multiple Resource Constraints," *Management Science*, 17(12):B803-B816 (August 1971).
19. Deckro, Richard F. and John E. Hebert. "Resource Constrained Project Crashing," *OMEGA International Journal of Management Science*, 17(1):69-79 (1989).
20. Deckro, Richard F., E. P. Winkofsky, John E. Hebert, and Roger Gagnon. "A Decomposition Approach to Multi-Project Scheduling," *European Journal of Operational Research*, 51:110-118 (1991).
21. Demeulemeester, E. "Minimizing Resource Availability Costs in Time-Limited Project Networks," *Management Science*, 41(10):1590-1598 (October 1995).
22. Demeulemeester, E., B. Dodin, and W. Herroelen. "A Random Activity Network Generator," *Operations Research*, 41(5):972-980 (September-October 1993).
23. Demeulemeester, E. and W. Herroelen. "A Branch-and-Bound Procedure for the Multiple Resource-Constrained Project Scheduling Problem," *Management Science*, 38(12):1803-1818 (December 1992).
24. Demeulemeester, E., W. Herroelen, W. P. Simpson, S. Baroum, J. H. Patterson, and Kum-Khiong Yang. "On a Paper by Christofides et al. for Solving the Multiple-Resource Constrained, Single Project Scheduling Problem," *European Journal of Operational Research*, 76(1):218-228 (July 1994).
25. Demeulemeester, E. L. and W. S. Herroelen. "An Efficient Optimal Solution Procedure for the Preemptive Resource-Constrained Project Scheduling Problem," *European Journal of Operational Research*, 90(2):334-348 (April 1996).
26. Fogarty, T. *Implementing the Genetic Algorithm on Transputer Based Parallel Processing Systems*, chapter in *Parallel Problem Solving from Nature* (H. P. Schwefel and R. Männer, editors), 145-149. Springer-Verlag, Berlin, 1991.
27. Foldes, Stephan and François Soumis. "Pert and Crashing Revisited: Mathematical Generalizations," *European Journal of Operational Research*, 64(2):286-294 (January 1994).
28. Franke, G. "Zur Ablaufplanung in Netzwerken (Scheduling in Networks)," *Z. für Betriebswirtschaftliche Forschung*, 23(106) (1971).
29. Gen, Mitsuo and Runwei Cheng. *Genetic Algorithms and Engineering Design*. New York: John Wiley and Sons, 1997.
30. Goldberg, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading MA: Addison-Wesley, 1989.

31. Gorges-Schleuter, M. *Explicit Parallelism of Genetic Algorithms Through Population Structures*, chapter in *Parallel Problem Solving from Nature* (H. P. Schwefel and R. Männer, editors), 150–159. Springer-Verlag, Berlin, 1991.
32. Grefenstette, John J. *Incorporating Problem Specific Knowledge into Genetic Algorithms*, chapter in *Genetic Algorithms and Simulated Annealing* (L. Davis, editor), 42–60. Pitman, London, 1987.
33. Hall, Nicholas G. and Marc E. Posner. *Generating Experimental Data for Scheduling Problems*. Working Paper, Ohio State University, December 1996.
34. Hamacher, Horst W. “A Note on K-Best Network Flows,” *Annals of Operations Research*, 57:65–72 (1995).
35. Hartmann, Sönke. *Project Scheduling with Multiple Modes: A Genetic Algorithm*. Manuskripte aus den Instituten für Betriebswirtschaftslehre, No. 435, University of Kiel, Germany, March 1997.
36. Herroelen, Willy S. “Resource-Constrained Project Scheduling—The State of the Art,” *Operational Research Quarterly*, 23(3):261–275 (1972).
37. Ho, James K., Tak C. Lee, and R. P. Sundarraj. “Decomposition of Linear Programs Using Parallel Computation,” *Mathematical Programming*, 42:391–405 (1988).
38. Hughes, David. “Advanced USAF Mission Planning System will Serve Fighters, Bombers and Transports,” *Aviation Week and Space Technology*, 134:52–53 (June 1991).
39. Hughes, David. “New Planning Software Aids Bosnian Airdrops,” *Aviation Week and Space Technology*, 138(17):59–61 (April 1993).
40. Hughes, David. “Navy System to Halve Strike Planning Time,” *Aviation Week and Space Technology*, 142:48–49 (March 1995).
41. Hurley, Marcus. “JFACC, Taking the Next Step,” *Joint Force Quarterly*, 7:60–65 (Spring 1995).
42. Ibaraki, T. “Enumerative Approaches to Combinatorial Optimization,” *Annals of Operations Research*, 11(1-4) (January 1988).
43. Jackson, Richard H. F., Paul T. Boggs, Stephen G. Nash, and Susan Powell. “Guidelines for Reporting Results of Computational Experiments. Report of the ad hoc Committee,” *Mathematical Programming*, 49:413–425 (1991).
44. Jensen, Paul A., “Integer Programming.” Integer Programming course handout. School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, OH, July 1994.
45. Kelley, James E. “Critical-Path Planning and Scheduling: Mathematical Basis,” *Operations Research*, 9(3):296–320 (May-June 1961).
46. Kohlmorgen, Udo, Hartmut Schmeck, and Knut Haase. *Experiences with Fine-Grained Parallel Genetic Algorithms*. Working Paper, University of Karlsruhe, D-76128 Karlsruhe, December 1996.



47. Kolisch, Rainer and Thomas Frase. "Minimizing Resource Costs When Meeting Tight Deadlines in a Project Environment." Unpublished Report, 1996.
48. Kolisch, Rainer and Arno Sprecher. "PSPLIB — A Project Scheduling Problem Library," *European Journal of Operational Research*, 96:205–216 (1996).
49. Kolisch, Rainer and Arno Sprecher. *PSPLIB — A Project Scheduling Problem Library*. Manuskripte aus den Instituten für Betriebswirtschaftslehre, No. 396, University of Kiel, Germany, March 1996.
50. Kolisch, Rainer, Arno Sprecher, and Andreas Drexel. *Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems*. Manuskripte aus den Instituten für Betriebswirtschaftslehre, No. 301, University of Kiel, Germany, December 1992.
51. Kolisch, Rainer, Arno Sprecher, and Andreas Drexel. "Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems," *Management Science*, 41(11):1693–1703 (1995).
52. Kronsjö, Lydia and Dean Shumsheruddin, editors. *Advances in Parallel Algorithms*. New York: Halsted Press, 1992.
53. Kumar, Vipin, Ananth Grama, Anshul Gupta, and George Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Redwood City, California: Benjamin/Cummings Publishing, 1994.
54. Lai, Ten-Hwang and Alan Sprague. "Performance of Parallel Branch-and-Bound Algorithms," *IEEE Transactions on Computers*, c-34(10):962–964 (October 1985).
55. Lamont, Gary B., George H. Gates, and Scott M. Brown. *Genetic Algorithms Combined with Deterministic Search*. Working Paper, Air Force Institute of Technology, Wright-Patterson AFB Ohio, January 1997.
56. Lasdon, Leon S. *Optimization Theory for Large Systems*. New York: MacMillan Publishing, 1970.
57. Lawler, Eugene L. "A Procedure for Computing the K-Best Solutions to Discrete Optimization Problems and its Application to the Shortest Path Problem," *Management Science*, 18(7):401–405 (March 1972).
58. Lee, Jae-Kwan and Yeong-Dae Kim. "Search Heuristics for Resource Constrained Project Scheduling," *Journal of the Operational Research Society*, 47:678–689 (1996).
59. Li, Guo-Jie and Benjamin W. Wah. "Coping with Anomalies in Parallel Branch-and-Bound Algorithms," *IEEE Transactions on Computers*, c-35(6):568–573 (June 1986).
60. Manderick, Bernard and Piet Spiessens. "Fine-Grained Parallel Genetic Algorithms." *Proceedings of the Third International Conference on Genetic Algorithms*, edited by Schaffer, J. D. 428–433. Morgan Kaufmann, San Mateo California, 1989.
61. Maybury, Mark T. *Distributed, Collaborative, Knowledge Based Air Campaign Planning*, chapter in AGARD Lecture Series 200: Knowledge Based Functions in Aerospace Systems. France: Advisory Group for Aerospace Research and Development, November 1995.

62. Michalewicz, Zbigniew. *Genetic Algorithms + Data Structures = Evolution Programs*. Berlin: Springer-Verlag, 1996.
63. Murty, Katta G. "An Algorithm for Ranking All the Assignments in Increasing Order of Cost," *Operations Research*, 16:682-687 (1968).
64. Nauss, Robert M. *Parametric Integer Programming*. Missouri: University of Missouri Press, 1979.
65. "Common Hardware and Software for the US Army's ATCCS," *International Defense Review*, 22(5):86-88 (May 1989).
66. Özdamar, Linet. *A Genetic Algorithm Approach to a General Category Project Scheduling Problem*. Research Report, Marmara University, Istanbul, 1996.
67. Özdamar, Linet. "A Genetic Algorithm Approach to a General Category Project Scheduling Problem." (to appear in *IEEE Transactions* in November 1998), July 1998.
68. Özdamar, Linet and Gündüz Ulusoy. "A Local Constraint Based Analysis Approach to Project Scheduling under General Resource Constraints," *European Journal of Operational Research*, 79:287-298 (1994).
69. Özdamar, Linet and Gündüz Ulusoy. "A Framework for an Interactive Project Scheduling System under Limited Resources," *European Journal of Operational Research*, 90:362-375 (1996).
70. Patterson, James H. "A Comparison of Exact Approaches for Solving the Multiple Constrained Resource, Project Scheduling Problem," *Management Science*, 30(7):854-867 (July 1984).
71. Petrovic, R. "Optimization of Resource Allocation in Project Planning," *Operations Research*, 16:559-586 (1968).
72. Pinedo, Michael. *Scheduling: Theory, Algorithms, and Systems*. New Jersey: Prentice Hall, 1995.
73. Press, Barry. "The US Air Force TEMPLAR Project Status and Outlook." *IEEE Western Conference on Knowledge Based Engineering and Expert Systems*. 42-48. 1986.
74. Pritsker, Alan B., Lawrence J. Watters, and Phillip M. Wolfe. "Multi-Project Scheduling with Limited Resources: a Zero-One Programming Approach," *Management Science*, 16(1):93-108 (September 1969).
75. Riestler, W. F. and R. Schwinn. "Projekplanungsmodelle (Project Planning Models)," *Physica, Würzburg-Wien* (1970).
76. Rosen, J. B. and Robert S. Maier. "Parallel Solutions of Large-Scale, Block-Angular Linear Programs," *Annals of Operations Research*, 22:23-41 (1990).
77. Simpson, David. "CSC Marches from Cold War to Commercial Wars," *Systems Integration Business*, 25(7):22-27 (July 1992).
78. Slowiński, Roman, Boleslaw Soniewicki, and Jan Węglarz. "DSS for Multiobjective Project Scheduling," *European Journal of Operational Research*, 79:220-229 (1994).
79. Sprecher, Arno. "Resource-Constrained Project Scheduling: Exact Methods for the Multi-Mode Case," *Springer-Verlag* (1994).

80. Sprecher, Arno and Andreas Drexl. *Solving Multi-Mode Resource-Constrained Project Scheduling Problems by a Simple, General and Powerful Sequencing Algorithm. Part I: Theory*. Manuskripte aus den Instituten für Betriebswirtschaftslehre, No. 385, University of Kiel, Germany, January 1996.
81. Sprecher, Arno and Andreas Drexl. *Solving Multi-Mode Resource-Constrained Project Scheduling Problems by a Simple, General and Powerful Sequencing Algorithm. Part II: Computation*. Manuskripte aus den Instituten für Betriebswirtschaftslehre, No. 386, University of Kiel, Germany, January 1996.
82. Sprecher, Arno, Rainer Kolisch, and Andreas Drexl. "Semi-Active, Active, and Non-Delay Schedules for the Resource-Constrained Project Scheduling Problem," *European Journal of Operational Research*, 80:94-102 (1995).
83. Stinson, Joel P., Edward W. Davis, and Basheer M. Khumawala. "Multiple Resource-Constrained Scheduling Using Branch and Bound," *AIIE Transactions*, 10(3):252-259 (September 1978).
84. Sweeney, Dennis J. and Richard A. Murphy. "A Method of Decomposition for Integer Programs," *Operations Research*, 27(6):1128-1141 (November 1979).
85. Sweeney, Dennis J. and Richard A. Murphy. "Branch and Bound Methods for Multi-Item Scheduling," *Operations Research*, 29(5):853-864 (September-October 1981).
86. Talbot, F. Brian. *An Integer Programming Algorithm for the Resource-Constrained Project Scheduling Problem*. PhD dissertation, Pennsylvania State University, 1976.
87. Talbot, F. Brian. "Resource-Constrained Project Scheduling with Time-Resource Tradeoffs: The Nonpreemptive Case," *Management Science*, 28:1197-1210 (1982).
88. Talbot, F. Brian and James H. Patterson. "An Efficient Integer Programming Algorithm with Network Cuts for Solving Resource-Constrained Scheduling Problems," *Management Science*, 24(11):1163-1175 (July 1978).
89. Tanese, Reiko. "Distributed Genetic Algorithms." *Proceedings of the Third International Conference on Genetic Algorithms*, edited by Schaffer, J. D. 434-439. Morgan Kaufmann, San Mateo California, 1989.
90. Tomlin, J. A. *Branch and Bound Methods for Integer and Nonconvex Programming*, chapter in *Integer and Nonlinear Programming*. Amsterdam: North-Holland Publishing, 1970.
91. Vercellis, Carlo. "Constrained Multi-Project Planning Problems: A Lagrangean Decomposition Approach," *European Journal of Operational Research*, 78:267-275 (1994).
92. Wiest, Jerome D. and Ferdinand K. Levy. *A Management Guide to PERT/CPM: with GERT/PDM/DCPM and other Networks* (Second Edition). New Jersey: Prentice-Hall, 1977.
93. Wiley, Victor D. *Optimization Analysis for Design and Planning of Multi-Project Programs*. MS thesis, AFIT/GOR/ENS/96M-18. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB Ohio, March 1996.

94. Wiley, Victor D., Richard F. Deckro, and Jack A. Jackson. "Optimization Analysis for Design and Planning of Multi-Project Programs," *European Journal of Operational Research*, 107(2):492-506 (June 1998).
95. Wu, Youfeng and Ted G. Lewis. "Parallel Algorithms for Decomposable Linear Programs." *1990 International Conference on Parallel Processing*. 27-34. 1990.

### *Vita*

Captain John C. Van Hove was born on 14 May 1967 in Berlin, New Jersey. In 1985, he graduated from Buena Regional High School in Buena, New Jersey, and went on to attend the United States Air Force Academy. He graduated from the Academy in June 1989 with a Bachelor of Science in Operations Research. Upon graduation, he received a regular commission in the United States Air Force and served his first tour of duty with Rome Laboratory at Griffiss AFB, New York. He began as a Neural Systems Analyst, performing research in the field of neural networks with a team serving as the lab's artificial neural systems focal point. He went on to work as the lab's Advanced Planning System Lead Engineer where he led the development, testing, and fielding of a multi-million dollar force level air combat planning tool. In August 1993, he entered the School of Engineering, Air Force Institute of Technology (AFIT). In 1995, he received a Master of Science in Operations Research and stayed on at AFIT to work on his Doctorate.

Permanent address: 5521 Honeyleaf Way  
Dayton, Ohio 45424

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE September 1998	3. REPORT TYPE AND DATES COVERED Doctoral Dissertation		
4. TITLE AND SUBTITLE AN INTEGER PROGRAM DECOMPOSITION APPROACH TO COMBAT PLANNING		5. FUNDING NUMBERS		
6. AUTHOR(S) John C. Van Hove, Captain, USAF				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) AFIT/ENS Bldg. 640 2950 P Street WPAFB, OH 45433-7765		8. PERFORMING ORGANIZATION REPORT NUMBER  AFIT/DS/ENS/98-1		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Dr. Neal Glassman AFOSR/NM 110 Duncan Ave. Ste B115 Bolling, AFB DC 20332-8080 202-767-5026 DSN 297-5026		10. SPONSORING/MONITORING AGENCY REPORT NUMBER		
11. SUPPLEMENTARY NOTES				
12a. DISTRIBUTION AVAILABILITY STATEMENT Approved for public release; distribution unlimited		12b. DISTRIBUTION CODE		
13. ABSTRACT (Maximum 200 words) <p>Over the last two decades, our military forces have been working to incorporate the latest computer technology into the combat planning process. The earliest efforts use word processors, spreadsheets, and databases to organize planning data and to display high level summaries for commanders. Later efforts perform feasibility checks as missions are planned to insure that the necessary resources are available and that the assets requested are capable of meeting the assigned scheduling requirements. Some of the most recent computer planning tools have included the capability to automatically plan individual missions or groups of missions. These automated efforts have been heuristic in nature due to the time limitations inherent to real-time combat planning. The methodologies in this research offer effective optimal alternatives to the limited heuristics available in the current planning tools. This research formulates and solves a new class of project scheduling problems with applications to both military and civilian planning. It is shown that the solution space for this class of problems may be reduced in order to improve the effectiveness of both optimal and heuristic solution methodologies. In addition, a general method for extending implicit enumeration algorithms to obtain k-best solution sets is developed. The reduced solution space and the general k-best solutions methodology are exploited to develop several efficient solution approaches for this new class of problems; an implicit enumeration algorithm, a decomposition approach, an evolutionary algorithm, and a hybrid decomposition approach. The applicability and flexibility of the methodology are demonstrated with a case study that focuses on the force level planning of combat missions for an air campaign.</p>				
14. SUBJECT TERMS Integer Program Decomposition; Combat Planning; Resource Constrained Project Scheduling		15. NUMBER OF PAGES 242		
		16. PRICE CODE		
17. SECURITY CLASSIFICATION OF REPORT  Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE  Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT  Unclassified	20. LIMITATION OF ABSTRACT  UL	